Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Systèmes automatiques

Présentée et soutenue par :
Thomas MOULARD

le : lundi 17 septembre 2012

Titre :

Optimisation numérique pour la robotique
et exécution de trajectoires référencées capteurs

Ecole doctorale :
Systèmes (EDSYS)

Unité de recherche :
LAAS-CNRS

Directeur(s) de Thèse :

Florent Lamiraux

Rapporteurs :

David Filliat
Thierry Géraud

Membre(s) du jury :
Philippe Souères
Rodolphe Gelin
David Filliat
Thierry Géraud
Florent Lamiraux

Table des matières

Remerciements

石の上にも三年

Proverbe japonais

J'aimerais tout d'abord remercier David et Théo de bien avoir voulu relire mon manuscrit et de m'avoir fait part de leurs remarques. Je remercie également Rodolphe Gelin d'avoir accepté de faire partie de mon jury de thèse.

Cette thèse est l'aboutissement d'un parcours académique où de nombreux chemins de travers auront été empruntés. Notamment, je remercie toute l'équipe du LRDE et en particulier Akim et Théo de m'avoir initié à la recherche. Sans eux, il m'est évident que ces dernières années auraient eu une autre couleur. Une pensée particulière également pour mes collègues de GOSTAI et à Jean-Christophe Baillie pour m'avoir amené à la robotique et, à son corps défendant (!), à cette thèse. Jean-Paul a ensuite su trouver les mots pour, en quelques heures, expliquer ce qui me paraissait alors d'une complexité insondable. Arrivé à Tsukuba, Florent, Pierre-Brice et Olivier ont pris le temps de m'expliquer la planification de mouvement et l'optimisation numérique, bien au-delà de leur rôle d'encadrants. Un second merci à Florent qui est ensuite devenu mon directeur de thèse et qui possède cette faculté particulière de pouvoir transformer ce qui apparaît comme trois années difficiles en une expérience vraiment plaisante. Ma semaine au sein du groupe LAGADIC fut courte, mais enrichissante ! Merci à Eric Marchand pour l'introduction courte, et efficace, à la vision par ordinateur. Merci à Claire Dune pour ses conseils et sa disponibilité. Enfin, je remercie Eiichi et Abder pour les précieuses discussions que nous avons pu avoir, et qui risquent de continuer encore un peu !

Le JRL et Gepetto ont été deux groupes extraordinaires en terme de travail et d'ambiance. Un grand merci à tous les permanents pour leur bienveillance. Quant aux doctorants, merci au bureau des mèmes - Antonio, Maxime et Samory - en particulier, mais aussi à : Paul, François, Pierre, Karim, Mehdi, Nicolas, Nosan, Oussama, David, Sovan, Manish, Séb, Ali, Valentin, Duong et Layale. Mention spéciale aux jolis Gepettistes de cœur : Wassima et Wassim.

Ma gratitude revient à Yoshiko notre chère professeur qui nous a enseigné le japonais sans jamais se décourager ! Merci à Ayaka de prendre le temps de déchiffrer mes lettres en japonais, à Alice d'agiter les bras en l'air sans raison, à Martin pour les photos qu'il me donnera peut-être un jour, à Pauline d'avoir les yeux qui brillent à chaque fois qu'on lui montre un kanji, à Antonio et Maya

pour leur future invitation au Liban, à Mélanie pour avoir formé le meilleur binôme de canoë que l'on puisse imaginer, à Marine pour sa perspicacité et ses blagues, à Yuri pour cette incroyable semaine en Italie, à Julie et Anaïs pour les séances de psychologie de groupe, à Rika, Rie, Tomoko, Haruka, Takahiro, Waritta, Yukari, Kei, Keiko et Mariko. À Yukiko pour m'avoir accompagné durant cette dernière année et pour son aide à tous les niveaux... Mes collègues se joignent à moi pour la remercier de son attention constante quant au niveau de glycémie du groupe.

À mes バカ préférées, Yuka et Marion, pour ce noël inoubliable et tant d'autres choses encore.

À ma famille, à mes frères, et à mes parents qui ont poussé la dévotion pour leurs enfants à l'extrême limite de leurs capacités.

Publications

- T. Moulard, F. Lamiraux et O. Stasse. Trajectory Following for Legged Robots. In *International Conference on Biomedical Robotics and Biomechatronics (BioRob'2012)*, Rome, Italie, juin 2012.
- L. Baudouin, T. Moulard, N. Perrin, F. Lamiraux, O. Stasse et E. Yoshida. Real-time Replanning Using 3D Environment for Humanoid Robot. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS 2011)*, Bled, Slovénie, octobre 2011.
- T. Moulard. Using numerical optimization in path planning, application to humanoid robot walk planning. In *Journées Nationales de la Robotique Humanoïde*, Toulouse, France, avril 2011.
- T. Moulard. Coordinate Frames for Humanoids Robots. Ros Enhancement Proposal 120, novembre 2011. URL `http://www.ros.org/reps/rep-0120.html`

Contributions logicielles principales

- T. Moulard, F. Lamiraux, P.-B. Wieber et O. Stasse. RobOptim : un framework pour l'optimisation numérique en robotique. LGPL-3.0. URL `http://www.roboptim.net/`
- T. Moulard. ViSP Tracker : un composant robotique ROS pour le suivi d'objet en temps réel se fondant sur les algorithmes de la bibliothèque ViSP conçu par l'équipe LAGADIC. BSD. URL `http://ros.org/wiki/visp_tracker`
- T. Moulard. Motion Analysis Mocap : un composant robotique ROS pour le suivi temps réel d'objet par la capture de mouvements fondé sur le système Cortex de la société Motion Analysis. BSD. URL `http://ros.org/wiki/motion_analysis_mocap`
- T. Moulard. RCPDF : une description unifiée des zones de contact autorisées sur un robot. BSD. URL `http://ros.org/wiki/robot_contact_point`
- T. Moulard, David Lu. RobotModelPy : un module Python permettant le chargement de modèles au format URDF. BSD. URL `http://ros.org/wiki/robot_model_py`
- F. Keith, T. Moulard, Romeo : modèle du robot Romeo d'Aldebaran Robotics au format URDF. BSD. URL `http://ros.org/wiki/romeo`

1 Introduction

> It's difficult to be rigorous about whether a machine really "knows", "thinks", etc., because we're hard put to define these things. We understand human mental processes only slightly better than a fish understands swimming.
>
> John McCarthy

1.1 Les enjeux de la robotique

1.1.1 État de la robotique en 2012

Il a beaucoup été écrit, et il a beaucoup été dit sur la robotique. On annonce depuis plusieurs décennies son avènement prochain et pourtant elle peine à s'imposer dans nos quotidiens. Il y a pourtant des raisons d'espérer ! Le rapport PIPAME sur Le développement industriel futur de la robotique personnelle et de service en France (Erdyn Consultants, 2012) estime qu'en 2015, au niveau mondial, le marché de la robotique de service personnelle représentera 8 milliards de dollars tandis que le marché de la robotique de service industrielle représentera, quant à elle, 18 milliards de dollars. Les gouvernements investissent largement dans les programmes de recherche en Europe et aux États-Unis – la *National Robotics Initiative*, par exemple, est un programme de 70 millions de dollars pour la coopération humain/robot –. Une explication de l'intérêt grandissant des pouvoirs publics pour ces technologies est la nécessité, pour les pays développés, de trouver de nouveaux axes de croissance et de lutter contre la délocalisation. L'automatisation de l'industrie représente une stratégie, notamment mise en avant par le Symop – Syndicat des Entreprises de Technologies de Production – au travers de leur site "Robotcaliser" [1]. Cette large poussée en avant s'observe également par les compagnies telles iRobot qui ont réussi à faire un pas vers l'entrée de la robotique dans nos quotidiens. Leur robot aspirateur Roomba est un des exemples de produits robotiques grand public ayant trouvé son marché et commençant à concurrencer sérieusement les aspirateurs non autonome.

Une autre application susceptible d'accéder au stade industriel dans la prochaine décennie est la voiture autonome. Les défis proposés par la DARPA – Defense Advanced Research Projects Agency – fournissent un bon aperçu de la vitesse à laquelle la robotique peut progresser : en 2004, aucune voiture n'a réussi à réalise plus de 11,78 km sur les 240 km que comptait la course. En 2005, cinq véhicules ont terminé la course dans sa totalité, l'équipe la plus rapide étant celle de Stanford avec un temps de parcours total de 6 heures et 54 minutes. En 2007, le défi a sensiblement changé puisque l'objectif était de réaliser 96 km de navigation autonome, en ville, tout en respectant le Code de la route et en prenant en compte la circulation extérieure : piétons et autres voitures non automatiques. Un aboutissement a également été, en 2012, la possibilité pour un aveugle d'être conduit par une voiture automatique conçue par Google de son domicile à un commerce local. Dans un domaine très lié à celui de la robotique, celui des capteurs, l'année 2010 a été l'année de la Kinect. Ce capteur destiné à la console de jeux Xbox 360 de Microsoft filme l'utilisateur tout en construisant simultanément une carte de profondeur dense. Cet accessoire permet de jouer aux jeux sans manette, en utilisant son propre corps pour interagir avec la console. La conception de ce capteur a donné à la communauté robotique un moyen peu onéreux – 150 euros environ – de percevoir l'environnement en 3D. Un capteur aux capacités proches, mais dédié à la recherche scientifique tel que le Swiss Ranger est actuellement vendu pour plusieurs milliers d'euros selon les modèles. Un tel changement dans l'échelle des prix, rendu possible par une industrialisation de masse affecte évidemment de manière indirecte le domaine.

1. Site officiel : `http://www.robotcaliser.com/`

FIGURE 1.1 – Le robot PR2 de la société Willow Garage.

Le gain en maturité du marché de la robotique se voit également dans les laboratoires de recherche. Il y a encore dix ans, les robots présents dans les laboratoires étant dans la quasi-totalité des cas réalisés par les chercheurs sur place. Cela nécessitait des compétences très étendues de la mécanique à l'électronique afin de réaliser une intégration correcte des différents composants. Comme tous les prototypes, ces robots étaient généralement peu fiables et difficiles à utiliser. Depuis quelques années la tendance s'inverse et les robots des laboratoire de recherche sont en train de devenir des "produits finis" manufacturés par quelques grands groupes industriels tel que Kuka en Allemagne ou Kawada Industries au Japon ou bien encore par des start-ups innovantes comme Willow Garage aux États-Unis. Ce dernier exemple est révélateur du niveau de qualité atteint par la robotique mobile à roue : le robot PR2, voire Figure 1.1, est certes cher – 400 000 dollars ! – mais comporte 7 caméras, un projecteur de lumière structurée, deux bras perfectionnés, deux capteurs lasers permettant de cartographier l'environnement, deux ordinateurs puissants et un ensemble de roues motorisées permettant un déplacement omnidirectionnel. Un tel robot arrive avec un ensemble de logiciels préinstallés permettant de s'abstraire d'une large partie des problèmes robotiques "de base" : calibration des caméras, cartographie et navigation automatique, génération de trajectoires afin de réaliser des scénarii type "pick and place" où le robot doit prendre un objet et le déposer ailleurs. Posséder un robot ayant un tel niveau de fonctionnalité dès "qu'on le sort de sa boîte" est un signe important et n'était pas envisageable ne serait ce qu'il y a cinq ans. De plus, comme tout produit construit en série, aussi petite soit-elle, ce robot, le PR2, bénéficie de tests de fiabilité, d'une documentation importante et d'une véritable assitance technique. Le gain en productivité pour les chercheurs

FIGURE 1.2 – Les robots HRP-4, HRP-2, HRP-3 et HRP-4c (de gauche à droite).

est réel et dès lors, il est clair que la robotique mobile est en train de passer d'une activité de recherche composée de prototypes à une activité industrielle structurée par des acteurs imposant leur plate-forme logicielle et matérielle.

1.1.2 L'enfance de l'Art de la robotique humanoïde

La robotique humanoïde, quant à elle, se démarque des autres robots mentionnés plus tôt. Alors que les drones et autres robots mobiles commencent à s'enorgueillir d'une qualité quasi industrielle, les robots humanoïdes sont encore au stade de l'enfance. Tant la conception mécanique, que les systèmes de contrôle ou bien encore des systèmes de perception adaptés restent à améliorer pour arriver à obtenir un robot pouvant évoluer de manière réellement autonome. Poussées par la fiction d'une part et la comparaison à l'homme d'autre part, les attentes pour ce type de technologies sont importantes alors que les résultats pratiques restent difficiles à obtenir. L'instabilité inhérente à la locomotion bipède et le nombre important de degrés de liberté – de possibilité de mouvement – rendent les schémas de contrôle particulièrement difficiles à concevoir. De nombreuses questions restent ouvertes telles que : comment réagir à une perturbation extérieure pendant la marche – le robot glisse ou bien il est poussé – ? Quelle conception mécanique permettrait d'assurer à la fois une résistance aux impacts, pour la course, le saut, mais également une grande vélocité pour d'autres usages, comme taper dans un ballon ? Quels sont les modèles optimaux pour générer des mouvements stables le plus rapidement possible ? Répondre à toutes ces questions, qui ne sont pas uniquement des questions algorithmiques, est primordial pour arriver à concevoir un robot humanoïde utile et autonome.

Les robots humanoïdes se divisent en deux catégories : ceux de petite taille comme le robot Nao d'Aldebaran Robotics (Wikipedia, 2011) et les humanoïdes de grande taille tels que ceux développés dans le cadre du projet HRP ou bien

encore par les sociétés avec le robot Partner ou Honda avec robot Asimo. Les problématiques posées par les deux types d'humanoïde sont assez différentes. Les robots de petite taille utilisent des actionneurs moins puissants, de qualité moindre, sont d'une conception moins précise par ils utilisent souvent des squelettes en plastique plutôt qu'en métal pour des questions de coût. Leurs capacités de calcul sont également souvent très limitées au point qu'un ordinateur externe est souvent nécessaire pour les commander. Les enjeux sont donc ici comment limiter les calculs nécessaires à bord du robot étant donné les fortes contraintes sur la capacité de calcul, comment prendre en considération les imperfections dans la modélisation du robot, la flexibilité de ses différentes parties ou bien encore la mauvaise précision de ses actionneurs. L'objectif étant de fournir à moyen terme un robot compagnon réalisant des tâches simples comme jouer avec un humain ou fournir des services simples comme la télésurveillance. Les capacités de manipulation de ces humanoïdes sont, à l'heure actuelle, très limitées, voire inexisantes.

Au contraire, les grands humanoïdes sont extrêmement onéreux et composés de pièces usinées très précisément et d'actionneurs extrêmement performants. De ce fait, les problèmes d'identification de modèles ne se posent qu'à la marge et le mouvement réel du robot est très proche de sa simulation faisant l'hypothèse de mouvements de corps rigides. Les capacités de calcul sont également plus importantes bien qu'inférieures à ce que l'on peut trouver dans un robot mobile où un poids important ne pose pas de problème particulier. Selon les modèles, ces robots peuvent réaliser des tâches de manipulation complexe. Les défis posés par ces robots sont différents : ils sont plus lourds, plus grands et donc toute chute leur est généralement fatale. La commande à gain fort courante sur ce type de système rend également le comportement du système potentiellement dangereux : pour atteindre sa consigne, le robot aura tendance à appliquer le couple maximal de ses actionneurs, quitte à s'endommager lui-même ou à blesser un humain se trouvant à proximité. Ils restent donc, à l'heure actuelle, des outils de recherches dangereux impropres à une utilisation à proximité d'humains.

1.1.3 Le projet japonais HRP

La totalité des recherches effectuées durant cette thèse et qui sont décrites ici ont été validées sur la plate-forme robotique HRP-2 (Kaneko et collab., 2004) du LAAS. Une version alternative est illustré par la Figure 1.3. Le robot humanoïde HRP-2 "Promet" est un robot humanoïde japonais mesurant 154cm et pesant 58kg. Il dispose de deux jambes à six degrés de liberté, de deux bras à six degrés et de deux mains à un degré de liberté commandant l'ouverture de la pince composant chaque préhenseur. Deux degrés de liberté sont affectés au mouvement de la tête et les deux derniers actionnent l'articulation du torse du robot. Cette dernière étant une caractéristique unique de cette plate-forme. Le nombre total de degrés de liberté du système est donc de 30. Les capteurs de ce robot comprennent quatre capteurs de force six-axes aux chevilles et au poignet, ainsi qu'une centrale inertielle dans le torse mesurant la vitesse angulaire et l'accélération linéaire du torse. Un système de caméras composé de deux paires stéréo situées dans la tête est également présent. Une première paire possède des objectifs grand-angles permettant de percevoir une large portion de l'environnement entourant le robot tandis que la seconde a un angle de vue plus étroit, permettant une bonne précision lors de la manipulation. Ce robot dis-

FIGURE 1.3 – Le robot HRP-2 réalisant une tâche de manipulation.

pose d'un système absorbant les chocs dans chaque cheville afin de protéger la mécanique des impacts réalisés pendant la marche. Les traitements embarqués sont réalisés par deux ordinateurs reliés entre eux par un réseau Ethernet. La connexion avec l'extérieur est assurée par un point d'accès WiFi.

Ce robot conçu en 1998 a été suivi du robot HRP-3 en 2007, du robot HRP-4C en 2009 et du robot HRP-4 en 2010. HRP-3 s'est démarqué en étant le premier robot humanoïde résistant à l'eau ouvrant la porte à une utilisation de ces robots en extérieur. HRP-4c est un gynoïde, c'est-à-dire un robot humanoïde possédant l'apparence d'une femme. Les applications ciblent le domaine du mannequinat et du divertissement plus généralement. Le dernier robot de la série, HRP-4 est, quant à lui, plus léger tout en conservant sept degrés de liberté pour les bras et deux degrés de liberté pour les mains. Les différents modèles sont illustrés par la Figure 1.2.

1.2 Contributions, de la génération de mouvements à leur exécution

Le maître mot de ces trois ans et demi est finalement, et cela ne peut que se ressentir à la lecture de ce manuscrit, la polyvalence. Comme dans toute thèse expérimentale, il a été nécessaire de tenter de mettre en pratique des idées, de A à Z. C'est à la fois formateur et passionnant, mais m'a conduit à utiliser de très nombreux outils, algorithmes de l'État de l'Art et à les comprendre afin de

pouvoir les intégrer à l'approche que j'ai souhaité développer. Le travail réalisé étant principalement de faire communiquer les idées entre elles : comment un algorithme de vision peut-il servir la commande ? Comment les données capteur peuvent-elles être utilisées par différents composants ? Autant de questions qui, en terme d'ingénierie logicielle, consistent à lier des boîtes entre elles, encapsulant autant d'algorithmes et de stratégies de décision différentes. Il est toutefois difficile d'argumenter sur la pertinence des "flèches" sans expliquer au préalable ce que contiennent les "boîtes", d'autant que si certaines techniques sont classiques en robotique, d'autres sont issues des recherches du groupe GEPETTO et ne peuvent être considérées comme allant de soi. Le choix réalisé lors de la rédaction a donc été d'expliquer progressivement comment réaliser des scénarii complexes avec un robot humanoïde, de la théorie, à la pratique, mes contributions se situant davantage dans ce spectre. Le premier chapitre de ce manuscrit traite d'un problème particulier abordé dans le cadre de cette thèse : la représentation informatique des problèmes d'optimisation numérique. Ce problème peut sembler distinct de la robotique humanoïde, mais les techniques d'optimisation sont devenues un socle supportant tant d'algorithmes utiles qu'on ne peut pas éluder la question de la représentation de ces problèmes. L'objectif ici est de s'appuyer sur les caractéristiques des langages de programmation modernes, tel que le C++ afin de pouvoir définir un problème d'optimisation une fois puis de pouvoir le transmettre à différents solveurs de façon transparente. Une application robotique simple valide cette approche, mais la contribution majeure reste la formulation et la modélisation informatique du problème. Le second chapitre explique pas à pas les techniques de génération de mouvement et d'exécution sur le robot. Ces techniques font partie de l'État de l'Art et il n'y a pas d'apport original à ce niveau. Cependant, le contrôleur construit sur ces techniques et permettant de suivre des trajectoires tout en incorporant les données capteur est un travail totalement original et sur lequel, à notre connaissance, aucun article antérieur n'a été publié. Le troisième chapitre est divisé en deux : tout d'abord la description informatique d'une pile de tâches asservie est un apport original. La suite du chapitre est dédiée à la localisation utilisant la vision sur un robot humanoïde. Cette partie n'est pas nouvelle en soi, car des résultats similaires ont déjà été publiés dans ce domaine, mais l'intégration de la localisation au schéma de contrôle proposé dans le Chapitre 3 est un travail original. Le dernier chapitre traite de l'intégration des différents composants sur notre plate-forme robotique. Dans ce chapitre, la totalité des algorithmes proposés font, soit partie de l'État de l'Art, soit ont été conçus par d'autres équipes partenaires du LAAS, notamment l'équipe LAGADIC de l'IRISA à Rennes. L'apport original de cette partie consiste à démontrer comment on peut construire une architecture robotique complète en utilisant des algorithmes existant afin d'atteindre des comportements de plus haut niveau.

2 Généricité et optimisation

> How do we convince people that in programming simplicity and clarity –in short: what mathematicians call “elegance”– are not a dispensable luxury, but a crucial matter that decides between success and failure?
>
> Edsger W. Dijkstra

2.1 Optimisation numérique et robotique

2.1.1 Introduction à l'optimisation numérique

La robotique s'appuie sur différents outils mathématiques pour résoudre les problèmes auxquels elle se trouve confrontée. Parmi ces outils, l'optimisation numérique figure à une place de choix.

L'optimisation numérique est une branche des mathématiques permettant de choisir automatiquement la meilleure solution parmi un ensemble de solutions possibles. Les applications possibles couvrent des domaines variés de la recherche opérationnelle aux statistiques et bien sûr la robotique. Dans ce domaine particulier, l'optimisation numérique permet par exemple d'optimiser un mouvement pour en améliorer certaines caractéristiques ou bien encore trouver la prochaine commande à envoyer aux moteurs du robot afin de réaliser une tâche donnée.

Résoudre un problème d'optimisation numérique signifie trouver les meilleurs paramètres résolvant le problème. Par meilleurs, il faut entendre les paramètres minimisant la valeur d'une fonction de coût. Intuitivement, on peut se représenter cette fonction comme un indicateur de la "qualité" de la solution trouvée. Une solution présentant un coût fort sera donc peu satisfaisante tandis qu'une solution présentant un coût faible sera, elle, très attrayante. Tout le raisonnement est ensuite fondé sur la différentiabilité de cette fonction de coût. Plus simplement, près d'une mauvaise solution, il n'y aura que des solutions légèrement meilleures et légèrement pires, mais jamais trop différentes. De ce fait, il "suffit" de chercher dans quelle direction aller pour se diriger vers les meilleures solutions pour finir par les trouver. Cette approche implique une limitation majeure : si la fonction de coût n'est pas convexe, on ne trouvera pas forcément la meilleure solution globale – parmi toutes les solutions – mais la meilleure solution locale. C'est-à-dire qu'au voisinage de cette solution, toutes les autres solutions sont de moins bonne qualité. Dans cette situation, tous les solveurs arrêtent leur recherche, mais rien ne garantit qu'ailleurs il n'existe pas une autre solution de meilleure qualité.

Ce raisonnement permet de trouver les meilleurs paramètres pour un problème donné, mais il est rare qu'une fonction de coût seule puisse capturer complètement un problème donné. En fonctionnant de manière indépendante du problème sous-jacent, on confère aux solveurs une grande polyvalence. Mais par là même, il devient nécessaire d'expliciter sous quelles conditions une solution au problème donné est acceptable. Prenons un exemple : pour aller d'un point A à un point B le plus rapidement possible, se déplacer avec une vitesse infinie est sans aucun doute la solution optimale. Cependant, une telle réponse ne présente guère d'intérêt dans la mesure où aucun système physique ne sera en mesure de l'exécuter.

Deux stratégies peuvent alors être choisies. La première est d'exprimer directement dans la fonction de coût cette information supplémentaire. Dans le cadre de l'exemple précédemment cité, on pourrait concevoir une fonction de coût réalisant la somme du temps de déplacement et de la vitesse moyenne du robot. De ce fait, une vitesse trop grande pénalise le résultat et par ce biais, le solveur tentera plutôt de transformer la trajectoire géométrique plutôt que de jouer sur la vitesse de déplacement du système. La limite de cette approche est

simple à comprendre : au fur et à mesure que les biais s'accumulent – le plus souvent en sommant les uns aux autres –, le coût initial peut se retrouver ignoré par le solveur, car numériquement négligeable. La fonction de coût n'a alors plus aucune justification physique et ses variations peuvent devenir de plus en plus difficiles à analyser pour le solveur. Une seconde solution consiste à ajouter des contraintes au problème. Ces contraintes sont exprimées sous la forme d'équations ou d'inéquations déterminant si une solution donnée est une solution acceptable. En robotique, des contraintes très courantes sont les bornes sur les butées articulaires. Il est rare sur un robot que les axes puissent se déplacer de manière totalement libre. De ce fait, il est courant d'ajouter des contraintes inégalités spécifiant que telle ou telle valeur encodant la position d'un degré de liberté doit rester dans un intervalle donné.

En exprimant comment résoudre un problème pratique via la construction d'une fonction de coût f et de contraintes $g_i > 0$, on **modélise** le problème afin qu'il soit soluble par optimisation numérique. Ce procédé n'est pas trivial, car la qualité de la modélisation impacte lourdement à la fois les temps de calcul et la qualité du résultat final.

2.1.2 Modélisation mathématique d'un problème

Résoudre un problème d'optimisation revient à trouver une solution pour :

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \text{ sous la contrainte } \mathbf{x} \in \mathbf{X} \tag{2.1}$$

où $f : \mathbb{R}^n \mapsto \mathbb{R}$ est la fonction de coût et $\mathbf{X} \subset \mathbb{R}^n$ est l'espace des solutions admissibles. Cet espace est habituellement défini par un ensemble de contraintes égalités et inégalités :

$$\mathbf{X} \equiv \begin{cases} c_i(x) = 0 & i \in \xi \\ c_j(x) \leq 0 & j \in \nu \end{cases} \tag{2.2}$$

c_i, c_j sont respectivement les ensembles de contraintes égalités et inégalités. i et j des indices identifiant les contraintes.

L'optimisation numérique a pour objectif de développer des stratégies – algorithmes – pouvant déterminer $\mathbf{x} \in \mathbf{X}$ minimisant les valeurs de la fonction f.

En absence d'argument fort tel que la convexité de f ainsi que des contraintes $c_i, i \in \xi \cup \nu$, la solution $\mathbf{x}$ fournie par le solveur peut être un minimum local .

En effet, la plupart des méthodes d'optimisation tentent de trouver un minimum en raffinant itérativement une solution à partir d'un candidat initial $\mathbf{x}_{\text{init}} \in \mathbf{X}$. Le solveur, dès lors, nécessite un critère d'arrêt à ce processus itératif. Ce critère est donné par les conditions de Karush-Kuhn-Tucker ou conditions KKT.

Définition 1. *Si $\mathbf{x} \in \mathbf{X}$ est un minimum local, alors il existe $\lambda_i, i \in \xi$ et $\mu_j, j \in \nu$ des constantes non nulles appelées multiplicateurs KKT tels que :*

Critère de stabilité

$$\nabla f(x) + \sum_{i\in\xi} \lambda_i c_i(x) + \sum_{j\in\nu} \mu_i c_i(x) = 0 \tag{2.3}$$

Classe de problèmes	*Fonction de coût*	*Contraintes*
Moindres carrés linéaires	$\sum_{i\in\{1,2,\ldots,n\}} (r_i - \mathbf{x}_i)^2$ où les r_i sont des constantes	non
LQP (opt. quadratique linéaire)	quadratique	linéaire
SQP (opt. non linéaire)	non linéaire	non linéaire

TABLE 2.1 – Zoologie des problèmes en optimisation numérique.

Critère de faisabilité primale

$$\begin{cases} c_i(x) = 0 & i \in \xi \\ c_j(x) \leq 0 & j \in \nu \end{cases} \tag{2.4}$$

Critère de faisabilité duale

$$\mu_j \geq 0, j \in \nu \tag{2.5}$$

Critère de complémentarité

$$\mu_j c_j(x) = 0, j \in \nu \tag{2.6}$$

Il est à noter que la Définition 1 fournit des conditions suffisantes afin de déterminer si un point est un minimum local. Sous conditions de régularité, et dans le cas où le problème est convexe, les conditions KKT sont nécessaires et suffisantes.

2.1.3 Zoologie des solveurs

Zoologie mathématique

Au-delà de cette présentation de la théorie générale, on peut regrouper les techniques de résolution en différentes catégories selon la difficulté des problèmes pouvant être traités. Nous ne nous attarderons ici que sur les techniques d'optimisation continues. Différentes caractéristiques peuvent rendre la résolution plus délicate :

- présence de contraintes,
- non convexité de la fonction de coût ou des contraintes,
- non linéarité de la fonction de coût ou des contraintes,
- fonction non – ou mal – définie en dehors des contraintes,
- Variations numériques fortes entraînantes des erreurs numériques difficiles à juguler,
- etc.

La combinaison de ces difficultés entraîne la classification des solveurs dans de très nombreuses catégories dont un extrait est détaillé dans le Tableau 2.1. Les algorithmes de résolution ont également des particularités intrinsèques qui peuvent être particulièrement intéressantes dans le cadre de la robotique. Par exemple, le démarrage à chaud ou "warm start" est la possibilité, pour un solveur, de résoudre plusieurs problèmes à la suite tout en se souvenant de son état

Logiciel	*Classe de problème*	*Caractéristiques*
C/C++ MinPack	Moindres carrés non linéaires	C/C++, réentrant, thread-safe
Coin IPOPT	SQP	C++, réentrant
CFSQP	SQP	C, réentrant

TABLE 2.2 – Zoologie des logiciels en optimisation numérique.

interne afin de pouvoir gagner en temps de calcul lorsque le nouveau problème est très proche du problème précédemment résolu. Cette situation intervient notamment lorsque l'on optimise une trajectoire tout en l'exécutant. L'optimisation se déroule pour différents instants proches dans le temps pour lesquels l'évolution de l'état du robot est minime. Une autre caractéristique est la capacité, pour un solveur, de pouvoir être interrompu à n'importe quelle itération tout en garantissant que la solution actuelle, bien qu'incomplète, respecte la totalité des contraintes du problème. Ce comportement est important lorsque l'optimisation est lancée dans un contexte où les contraintes temporelles sont fortes, telles que dans le contrôleur calculant les commandes du robot.

Zoologie logicielle

On l'aura compris : d'une théorie unique découle un ensemble d'algorithmes pouvant résoudre des problèmes de complexité variée. Plaçons-nous désormais à la place d'un roboticien devant résoudre un problème particulier. Quels logiciels peut-il utiliser pour ce faire ? Le Tableau 2.2 détaille une partie des paquets logiciels existants. Une conclusion s'impose au regard de cette liste de paquets logiciels : les techniques les plus compliquées à implémenter sont peu disponibles et il n'existe pas d'outil unifié permettant de résoudre différents types de problèmes. Enfin, chaque outil traite une catégorie de problème en particulier. Il est raisonnable de se demander si chaque utilisateur lorsqu'il commence à définir son problème est à même de choisir de manière pertinente le solveur – et donc le framework – adapté à son problème. Est-il certain que toutes les fonctions seront différentiables et continues deux fois ? Une fois ? Est-il certain de ne pas avoir besoin de contraintes inégalités, voire de ne pas avoir besoin du tout de contraintes ? Rien n'est moins sûr. Un exemple simple est la gestion des collisions. On peut tout à fait développer un outil robotique et le faire fonctionner dans un environnement ouvert avant de vouloir s'attaquer à un problème plus difficile et prendre en compte les collisions. Cela peut poser des problèmes extrêmement importants, car le solveur ne sera pas le même dans les deux cas. Cependant, il semble qu'avoir à réécrire, ou tout du moins adapter, le problème précédent à une nouvelle bibliothèque logicielle est inutile puisque finalement les deux problèmes se formalisent au sein d'un même paradigme.

2.2 De la difficulté à passer d'un paradigme unique à une implémentation unifiée

À première vue, il pourrait sembler que l'absence d'une architecture unifiée pour la résolution de problèmes d'optimisation numérique est le simple résultat d'un manque de volonté ou de coordination entre concepteurs. Nous allons montrer ici qu'il n'en est rien. Une simple agglomération des implémentations de différentes stratégies de résolution ne saurait donner un ensemble cohérent et performant en terme de temps de calcul sans une véritable réflexion sur comment modéliser le paradigme décrit dans la section 2.1 tout en prenant en compte les limites internes des langages de programmation. Nous allons tout particulièrement nous intéresser à la programmation orientée objet qui est la fondation d'une large proportion des langages modernes.

2.2.1 Typage, programmation orientée objet et familles de types

λ-calcul et théorie des types

Lors du développement de l'informatique, les langages de programmation ont tout d'abord été de simples outils permettant de représenter du code machine de manière succincte et plus facilement lisible par les humains. Cependant, avec la complexification croissante des applications, vérifier la correction des programmes est rapidement devenu un enjeu critique. Pour ce faire, les langages de programmation ont progressivement adopté le typage comme moyen de n'autoriser que la construction de programmes valides. Cette notion de type est issue de la théorie des types , une branche de la logique mathématique. La première théorie unifiée des types a été développée par Bertrand Russel au début du vingtième siècle dans les *Principia Mathematica* (Whitehead et Russel, 1910).

Afin de démontrer les difficultés que peuvent poser la modélisation d'un problème d'optimisation numérique, nous allons nous appuyer sur un système de calcul formel particulier, le λ-calcul inventé par Alonzo Church dans les années trente. Ce système de calcul formel a une expressivité équivalente à une machine de Türing. Nous allons tout d'abord nous concentrer sur la construction des lambda-termes , c'est-à-dire l'ensemble des expressions syntaxiquement correctes pouvant être construites en lambda-calcul. Ces lambda-termes peuvent être divisés en trois catégories : variables, applications et abstractions. Les variables : x, y, ... Les applications sont l'ensemble des expressions uv où u et v sont deux lambda-termes. Enfin les abstractions sont les expressions du type : $\lambda x.u$ où x est une variable et u un lambda-terme. Par exemple, l'application identité peut s'écrire en lambda-calcul : $\lambda x.x$.

La procédure qui, à partir de n'importe quel lambda-terme, tente de la simplifier en réalisant les applications par réécriture est appelée β-conversion . En réduisant un lambda-terme, on réalise le calcul associé à ce dernier. Une propriété utile serait de pouvoir déterminer si toutes les lambda-réductions terminent, c'est-à-dire que l'on peut réduire tous les lambda-termes jusqu'à un point où plus aucune réécriture n'est possible.

Hélas, ce n'est pas le cas en lambda-calcul. Soit :

$$\Delta \equiv \lambda x.x \tag{2.7}$$
$$\Omega \equiv \Delta\Delta \equiv (\lambda x.x)(\lambda x.x) \tag{2.8}$$

On peut remarquer que la β-reduction de Ω boucle indéfiniment :

$$(\lambda x.x)(\lambda x.x) =_\beta (\lambda x.x)(\lambda x.x) \tag{2.9}$$

En terme calculatoire, on peut considérer que ce programme boucle et ne termine jamais.

De ce constat, une autre interrogation se pose : peut-on construire un sous-ensemble des lambda-termes normalisables ? La réponse est oui, par l'utilisation du lambda-calcul simplement typé . Dans cette version du lambda-calcul, toute expression est annotée par un type. Les types sont de deux sortes : ι et $\tau_1 \to \tau_2$ à condition que τ_1 et τ_2 soient des types. ι représente le(s) type(s) primitif(s) du langage permettant de représenter les booléens ou un sous-ensemble des nombres entiers ou réels par exemple. Si x est de type ι, on a alors :

$$x \vdash \iota \tag{2.10}$$

... prononcé "ι type x". L'ensemble des types est quant à lui noté Γ.

En définissant des règles de typage appropriées, on peut restreindre les lambda-termes bien typés aux lambda-termes normalisables. Cependant, une limitation de cette approche est qu'une grande partie des expressions que l'on souhaiterait construire ne sont pas correctement typées. En particulier, il n'est pas possible de construire la fonction exponentielle dans ce paradigme.

Programmation orientée objet et typage

Indépendamment des systèmes de calculs formels, divers modèles plus pratiques ont été développés au cours de la seconde moitié du vingtième siècle. Notamment, la programmation orientée objet a été introduite par Ole-Johan Dahl et Kristen Nygaard dans Simula I au cours des années soixante. Ce langage avait pour but de permettre le développement de simulation à événements discrets. Ce paradigme de pensée s'est progressivement développé jusqu'à être massivement adopté par la majorité des langages de programmation au cours des années quatre-vingt-dix. L'adoption massive du C++ (Bjarne Stroustrup , 1983) a participé à la démocratisation de la POO.

La POO introduit la classe comme élément unitaire permettant la conception d'un programme informatique. Alors qu'auparavant, une séparation stricte était observée entre les données et les algorithmes, une classe est un élément du langage regroupant ces deux éléments. Les algorithmes fournis par une classe sont alors appelés "méthodes" tandis que les données prennent le nom "d'attributs". Cette réunification des données et des traitements sur les données s'inscrit pleinement dans la lignée de la vision informatique introduite par Von Neumann et l'architecture portant son nom. L'innovation clé de cette architecture a été de permettre la transmission sur un même bus des instructions composant un programme ainsi que des données sur lesquelles les instructions vont s'appliquer. On peut donc dire que la classe a poussé la réunification algorithmes/données jusqu'à la conception interne des logiciels.

Définition 2. *Un type peut soit, être un type fondamental noté ι, soit un type fonctionnel, soit un type de classe. En pratique, il existe plusieurs types fondamentaux : réels, entiers, booléens, mais le nombre de types de base ne change en rien les règles de typage.*

Définition 3. *Soit τ un type de classe. τ est défini par $a \in \mathfrak{A}_\tau$ l'ensemble des attributs – données – de la classe et $m \in \mathfrak{M}_\tau$ l'ensemble des méthodes de la classe.*

Notation : *on notera $\tau.a$ l'attribut a de la classe τ et $\tau.m$ la méthode m de la classe τ. Afin d'éviter les ambiguïtés, l'ensemble des symboles composant les éléments $\mathfrak{A}_\tau$ et $\mathfrak{M}_\tau$ sont disjoints.*

Définition 4. *Soit $\mathfrak{M}_\tau$ l'ensemble des méthodes d'une classe. Une méthode m du type de classe τ est une variable de type $\tau \to \tau'$ où τ' est un type quelconque.*

Notation : *L'application de la méthode m de la classe τ devrait s'écrire $a.m\tau$ où $a \vdash \tau$. Dans la mesure où l'on sait que $m \in \mathfrak{M}_\tau$, on peut se passer d'appliquer explicitement le premier argument du type fonctionnel et directement écrire $\tau.m$ pour signifier l'application de la méthode m.*

Définition 5. *Soit $\mathfrak{A}_\tau$ l'ensemble des attributs d'une classe. Un attribut a du type de classe τ est une variable de type τ' quelconque à l'exception des types fonctionnels prenant un type τ en argument.*

Exemple 1. *Soit A une classe définie par :*

$$A \equiv \{\underbrace{\{foo \vdash \iota\}}_{attribut}, \underbrace{\{bar \vdash A \to \iota, baz \vdash A \to \iota\}}_{méthodes}\} \tag{2.11}$$

A possède un attribut et deux arguments. D'après la définition de A et les Définitions 4 et 5, on a :

$$a \vdash A \Rightarrow a.foo \vdash \iota \tag{2.12}$$

$$a \vdash A \Rightarrow a.bar \vdash A \to \iota \tag{2.13}$$

Cette représentation offre une vue d'ensemble intégrant à la fois des données structurées et l'ensemble des opérations possibles sur ces dernières. Auparavant, la séparation de ces deux éléments rendait plus difficile la construction d'une vision globale : on ne pouvait, par définition, pas connaître la liste des opérations définies sur un groupe de données. Soit τ un type de classe, l'ensemble des fonctions prenant τ en entrée n'est pas connu lors de la compilation, car il peut être augmenté à tout moment d'un nouvel élément en définissant une nouvelle routine. Il convient donc de documenter la liste des opérations disponibles.

Inversement, il est souvent nécessaire d'augmenter le comportement d'une classe en y ajoutant de nouvelles données et/ou de nouveaux algorithmes. La POO modélise ce processus au travers de l'héritage de classe. Soit τ_1 un type de classe, τ_2 un type de classe héritant de τ_1. On dit également que τ_2 est un sous-type de τ_1, et est noté sous la forme :

$$\tau_2 <: \tau_1 \tag{2.14}$$

Cette notion, proche de celle d'inclusion en théorie des ensembles, permet de définir des familles de types interchangeables.

Définition 6. *Soit Γ un contexte de type. τ_1, τ_2, τ_3 trois types appartenant à ce contexte. La règle de typage des applications en programmation orientée objet est alors :*

$$u\ v \text{ bien formé} \Rightarrow u \vdash \tau_1 \to \tau_2 \wedge \tau_3 <: \tau_1 \tag{2.15}$$

Dès lors que τ_3 est un sous-type de τ_1, les fonctions prenant en argument une variable de type τ_1 acceptent également les arguments de type τ_3.

Définition 7. *Soit Γ un contexte de type. τ_1, τ_2 deux types.*
On dit que $\tau_2 <: \tau_1$ si et seulement si :

$$\mathfrak{A}_{\tau_1} \subset \mathfrak{A}_{\tau_2} \wedge \mathfrak{M}_{\tau_1} \subset \mathfrak{M}_{\tau_2} \tag{2.16}$$

Un sous-type se doit de contenir au moins tous les attributs et méthodes de la classe dont elle hérite afin d'interdire la réécriture vers des termes syntaxiquement faux. Le "principe de substitution de Liskov" (Liskov et Wing, 1994) définit, de manière équivalente, la relation de sous-typage sous l'énoncé suivant :

Définition 8. *Si $q(x)$ est une propriété démontrable pour tout objet x de type T, alors $q(y)$ est vraie pour tout objet y de type S tel que S est un sous-type de T.*

La Définition 7 introduit implicitement la notion de polymorphisme d'héritage. Le polymorphisme est la capacité pour un type fonctionnel de ne pas accepter en entrée un type unique, mais une famille de types. Le polymorphisme le plus simple pouvant être défini est la surcharge. Ce mécanisme autorise la création de fonctions se réécrivant vers des λ-termes différents selon le type de l'argument d'entrée. La formalisation de ce système est décrite dans (Castagna et collab., 1995).

Une autre forme de polymorphisme est le polymorphisme d'héritage. Si τ_1 et τ_2 sont deux types tels que $\tau_2 <: \tau_1$ et qu'il existe une méthode m dans τ_1 et τ_2, alors tout application de la méthode m à un objet de type τ_2 se réécrira vers le λ-terme $\tau_2.m$. Un sous-type peut donc fournir une réécriture alternative d'une fonction afin d'en spécialiser le comportement.

La généralisation de cette propriété à toutes les fonctions au lieu des méthodes uniquement permet d'obtenir le "multiple dispatch" (Muschevici et collab., 2008). Ce mécanisme est disponible dans un nombre assez limité de langages de programmation, dont Common Lisp. En effet, contrairement à la surcharge, le multiple dispatch ainsi que le polymorphisme d'héritage nécessite une résolution à l'exécution qui peut pénaliser l'implémentation des algorithmes. En limitant la résolution dynamique au type du premier argument des méthodes, on limite également le coût de la résolution à l'exécution. Ceci explique en partie l'une des raisons pour lesquelles le premier argument des méthodes est omis dans de nombreux langages de programmation. On "masque" par ce biais, le caractère spécifique de la résolution sur le premier argument de la méthode et on peut, une fois ce premier argument omis, considérer qu'il n'y a pas de résolution dynamique sur les arguments des méthodes.

Types paramétrés et génération de types

Face au coût induit par la résolution dynamique du polymorphisme par héritage, on peut se demander s'il existe un moyen d'arriver au même résultat sans

ralentir l'exécution d'un algorithme. Une solution est de générer des types afin de pouvoir déterminer à la compilation la résolution des fonctions polymorphes. Ce mécanisme est appelé "templates" en C++ et se formalisent en λ-calcul sous la forme de types paramétrés.

Définition 9. *Un type paramétré est une fonction d'ordre supérieur type $\Rightarrow$ type.*

Notation : *Soit τ un type paramétré. Le type τ paramétré par α est dénoté τ_α. Il représente l'application du type α à la fonction d'ordre supérieure τ.*

Les types paramétrés permettent de définir un nouveau type de polymorphisme, le polymorphisme paramétré. Ce polymorphisme peut se formaliser en rendant les méthodes des classes paramétrées dépendantes du paramètre de classe.

En pratique, ce type de polymorphisme est particulièrement intéressant, car il n'induit pas de coût à l'exécution.

2.2.2 Modélisation informatique du problème

La section 2.1 a introduit le formalisme et les outils que nous allons utiliser pour poser le problème de manière formelle.

Un solveur est un algorithme qui prend en entrée un problème d'optimisation et qui calcule un point dans l'espace des solutions. Nous allons noter ι le type primitif représentant une partie des réels et $[\iota]$ une liste de zéro, un ou plusieurs variables de type ι. τ_{prob} le type représentant le problème d'optimisation. Nous allons voir comment ce type est habituellement défini, introduire une paramétrisation alternative du problème fournie par RobOptim et enfin montrer quelles avancées cette nouvelle modélisation engendre. Nous allons travailler en utilisant un solveur générique qui sera dénoté :

$$\text{solv} \vdash \tau_{\text{prob}} \to \iota \tag{2.17}$$

La vision habituellement implémentée consiste à définir un unique type τ_{prob} dédié à un solveur précis. L'approche proposée ici consiste, à la place, à définir une famille de types formée par des classes, sous-classes éventuellement paramétrées pour tenter d'exprimer la totalité de la théorie de l'optimisation numérique sous la forme d'une et une seule modélisation.

De la fonction... au type fonctionnel ?

Naïvement, on pourrait penser qu'il n'y a rien de plus simple à modéliser qu'une fonction mathématique. En effet, les types fonctionnels ne sont-ils pas une transcription directe des fonctions mathématiques habituelles ? On aurait donc, à première vue :

$$\begin{array}{rcccc} f & : & \tau_1 & \to & \tau_2 \\ & & x & \mapsto & f(x) \end{array} \xhookrightarrow{\text{devient}} f \vdash \tau_1 \to \tau_2 \tag{2.18}$$

Cette modélisation convient, bien sûr, mais ne permet pas de ranger les catégories selon leur nature : continue, différentiable, etc.

En effet, pouvoir accéder au gradient de f est primordial, mais implémenter une fonction $\nabla(f)$ est difficile car dans de nombreux langages, dont celui utilisé

par RobOptim – C++ –, les fonctions ne sont pas des objets de premier ordre et leur manipulation est, de fait, mal-aisée.

Afin de pouvoir associer des métadonnées aux fonctions, une représentation naturelle est l'utilisation d'une hiérarchie de classe.

Le critère choisi est la classe de la fonction : $\mathcal{C}^0$, $\mathcal{C}^1$, ... à une nuance près. En effet, si une fonction de classe $\mathcal{C}^n$ est déclarée, elle doit non seulement être dérivable n fois et continue $n+1$ fois, mais les valeurs des n fonctions dérivées doivent pouvoir être calculées.

La classe de fonction associée à la possibilité de calculer les gradients est un facteur très important lors de l'évaluation du "niveau" de difficulté d'un problème d'optimisation. Dès lors, regrouper les fonctions selon ce critère semble être un choix raisonnable.

Définition 10. *Soit f une fonction mathématique continue dont on ne sait pas calculer la dérivée. Cette catégorie de fonctions est définie par la classe Fonction :*

$$Fonction \equiv \{\underbrace{\{\#entrée \vdash \iota\}}_{attributs}, \underbrace{\{calcule \vdash Fonction \to [\iota] \to \iota\}}_{méthode}\} \tag{2.19}$$

L'attribut #entrée la cardinalité de l'espace d'entrée de la fonction et la méthode calcule permet d'évaluer la fonction en un point. Nous ne traiterons ici que les fonctions dont l'espace de sortie est de cardinalité 1. En effet, les fonctions dont l'espace de sortie sont de cardinalité n peuvent être facilement représentables par n fonctions dont l'espace de sortie est de taille 1.

Définition 11. *Soit f une fonction mathématique continue et dont on sait calculer la dérivée une fois. Cette catégorie de fonctions est définie par la classe FonctionDérivable :*

$$FonctionDérivable \equiv \{\{\}, \{gradient \vdash FonctionDérivable \to [\iota] \to [\iota]\}\} <: Fonction \tag{2.20}$$

Le type FonctionDérivable hérite du type Fonction et fournit, de surcroît, la méthode gradient permettant d'évaluer le gradient de la fonction en un point.

Définition 12. *Soit f une fonction mathématique continue et dont on sait calculer la dérivée n fois. Cette catégorie de fonctions est définie par la famille de classe $Fonction_n$ définit par :*

$$Fonction_n \equiv \begin{cases} Fonction & si\ n = 0 \\ FonctionDérivable & si\ n = 1 \\ \{\{\}, \{gradient \vdash Fonction_n \to \iota \to [\iota] \to [\iota]\}\} <: Fonction_{n-1} & si\ n > 1 \end{cases} \tag{2.21}$$

On définit de sorte une famille de types formant une hiérarchie de classe sous la forme d'une chaîne où $\mathcal{C}^n <: \mathcal{C}^{n-1} <: \ldots <: \mathcal{C}^1 <: \mathcal{C}^0$.

À partir de $\mathcal{C}^2$, on remarquera que la méthode gradient est surchargée selon le nombre d'arguments. Avec un argument de type $[\iota]$, le gradient est évalué en un point. Avec deux arguments $o \vdash \iota$ et $\mathbf{x} \vdash [\iota]$, on aura pour $o \leq n$ le résultat de l'évaluation de la dérivée d'ordre o.

Cette hiérarchie de classe est le cœur de la modélisation informatique des problèmes d'optimisations, car elle permet d'exprimer à la fois la fonction de coût et les contraintes éventuellement présentes dans un problème. Il suffit alors à l'utilisateur souhaitant définir sa fonction F de la faire hériter du type adéquat de la famille Fonction_n.

Cette approche sous forme de hiérarchie linéaire de classes en forme de peignes a toutefois des inconvénients qui seront détaillés dans la section 2.4 de ce chapitre.

Des problèmes divers, un type unique

Nous avons donc toute une famille de types définissant des fonctions mathématiques : non seulement les Fonction_n qui modélisent les interfaces de notre programme, mais aussi tous les types correspondant aux fonctions présentes dans un problème d'optimisation particulier.

La question est désormais comment modéliser une "classe" de problème d'optimisation numérique de manière générique tout en assurant une sûreté maximum par typage.

La proposition réalisée ici est de définir une famille de types τ_{prob} paramétré par $n > 0$ type(s). Le premier définit le type de la fonction de coût tandis que tous les autres définissent les types de fonctions des contraintes.

Définition 13. *Soit $\tau_{prob(\tau_1,\tau_2,\tau_3,\dots)}$ tel que $\tau_i <:$ Fonction, $i \in \{1,\dots,n\}$:*

$$\begin{aligned}\tau_{prob(\tau_1,\tau_2,\tau_3,\dots)} \equiv \{\{&coût \vdash \tau_1,\\ &contraintes \vdash [\cup_{i\in\{2,\dots,n\}}\tau_i],\\ &bornes\_contraintes\_min \vdash [\iota],\\ &bornes\_contraintes\_max \vdash [\iota]\},\\ &\{\}\}\end{aligned} \tag{2.22}$$

Un problème d'optimisation est constitué d'une fonction de coût de type τ_1 et un ensemble de fonctions pouvant être de type τ_i, $i \in \{2,\dots,n\}$. Cet ensemble de fonctions associé à un ensemble d'intervalles permet de définir des contraintes égalité ou inégalité. Soit $contraintes_i$ la i-ème contrainte du problème, on a :

$$bornes\_contraintes\_min_i \leq contraintes_i \leq bornes\_contraintes\_max_i \tag{2.23}$$

On notera que construire une contrainte égalité revient à spécifier une borne minimum et maximum identique.

On remarquera que dans cette définition, le type $[\cup_{i\in\{2,\dots,n\}}\tau_i]$ est construit. Ce type est un "type union" et correspond à une variable qui peut contenir une instance d'un type parmi tous ceux listés dans l'union. Ce mécanisme est nécessaire car on ne sait pas à l'avance quelle sera le type de chaque contrainte. On est juste assuré que chaque instance fera partie de cet ensemble de types.

La Définition 13 permet donc de modéliser un problème d'optimisation numérique via l'utilisation d'un type paramétré générique. Cette modélisation couvre une large partie des problèmes d'optimisation avec ou sans contrainte tout en fournissant une interface unique. Nous sommes donc dans un cas où l'utilisateur peut définir son problème de manière naturelle, sans avoir à partir d'un solveur qui impose son formalisme spécifique.

Le chaînon manquant : le solveur

Évidemment, il ne saurait être question de terminer cette section sans revenir une dernière fois sur le cœur et l'âme du système : le solveur. Dans la section précédente, nous avons construit un type paramétré qui satisfait les contraintes énoncées au début de cette section, voire Équation 2.17.

En effet, on peut construire un type paramétré par le type de problème permettant de modéliser un solveur.

Définition 14. *Soit* $Solveur_\tau$ *une classe paramétrée par un type* τ *tel que* $\exists n, \exists \tau_1, \ldots, \tau_n, \tau <: Problème_{\tau_1,\ldots,\tau_n}$ *et défini par :*

$$Solveur_\tau \equiv \{\{\}, \{résous \vdash Solveur_\tau \to \tau \to [\iota]\}\} \tag{2.24}$$

La Définition 14 fournit la dernière famille de types indispensables à la modélisation d'un problème d'optimisation numérique. Nous allons désormais nous attacher à montrer que diverses classes de problèmes d'optimisation peuvent être exprimées via cette modélisation unique.

2.3 Exemples de formalisation de classes de problème

Afin de démontrer que le formalisme décrit dans la section précédente peut réellement permettre de décrire des problèmes d'optimisation de manière efficace et générique, nous allons construire les types représentant trois classes de problèmes en utilisant les familles de types décrites dans la section précédente. Les trois classes choisies sont :

- moindres carrés linéaires
- optimisation non linéaire : fonction de coût non linéaire, contraintes linéaires ou non linéaires. Le gradient des fonctions doit pouvoir être évalué.
- optimisation non linéaire : fonction de coût non linéaire, contraintes linéaires ou non linéaires. Le hessien des fonctions doit pouvoir être évalué (dérivée d'ordre 2).

2.3.1 Moindres carrés linéaires

Les moindres carrés linéaires utilisent une fonction de coût suivant une structure très particulière. Proposer un type modélisant cette structure est possible :

Définition 15. *Soit FonctionQuadratique un type tel que :*

$$FonctionQuadratique <: Fonction_2$$

Définition 16. *Soit FonctionLinéaire un type tel que :*

$$FonctionLinéaire <: FonctionQuadratique$$

Nous avons ici choisi de faire dériver les fonctions linéaires et quatratiques du type $\text{FonctionQuadrique}_2$. Cela ne correspond pas à la réalité mathématique dans la mesure où une fonction linéaire est infiniment dérivable, la dérivée étant nulle à partir de la seconde dérivation. Il n'est pas possible d'exprimer cet état de

fait au regard de la modélisation adoptée ce qui représente une limite du modèle de typage proposé. En pratique, les fonctions plus de deux fois dérivables ne présentent pas d'intérêt particulier et cette limite ne pose donc pas de problèmes en pratique.

Définition 17. *Soit MoindreCarré un type tel que*

$$\mathit{MoindreCarré} <: \mathit{FonctionQuadratique}$$

Pour rappel, la fonction s'exprime mathématiquement sous la forme :

$$\sum_{i\in\{1,2,\ldots,n\}} (r_i - \mathbf{x}_i)^2 \tag{2.25}$$

... où les r_i sont des constantes.

Cette fonction peut se modéliser par le type suivant :

$$\mathit{MoindreCarré} \equiv \{\{r \vdash [\iota]\}, \{\}\} <: \mathit{FonctionLinéaire} \tag{2.26}$$

où l'attribut r représente les r_i de l'équation précédente.

Une fois ces types définis, il ne reste plus qu'à instancier les types paramétrés avec un ensemble de paramètres adéquats.

Exemple 2. *Paramétrisation proposée :*
Fonction de coût *MoindreCarré*
Contraintes *non*
Problème $\mathit{Problème}_{prob(\mathit{MoindreCarré})}$

2.3.2 Optimisation non linéaire (gradient uniquement)

Nous allons traiter ici la classe de problèmes suivante : la fonction de coût est non linéaire et les contraintes séparées en deux types distincts : linéaires ou non linéaires.

Exemple 3. *Afin de pouvoir décrire cette classe de problème, la paramétrisation des types choisie est la suivante :*
Fonction de coût $\mathit{Fonction}_1$
Contraintes linéaires *FonctionLinéaire*
Contraintes non linéaires $\mathit{Fonction}_1$
Problème $\mathit{Problème}_{prob(\mathit{Fonction}_1, \mathit{FonctionLinéaire}, \mathit{Fonction}_1)}$

2.3.3 Optimisation non linéaire (gradient et hessien)

Si l'on définit la même classe de problème, mais en ajoutant la possibilité d'évaluer les hessiens des fonctions non linéaires, on obtient la paramétrisation suivante.

Exemple 4. *Paramétrisation proposée :*
Fonction de coût $\mathit{Fonction}_2$
Contraintes linéaires *FonctionLinéaire*
Contraintes non linéaires $\mathit{Fonction}_2$
Problème $\mathit{Problème}_{prob(\mathit{Fonction}_2, \mathit{FonctionLinéaire}, \mathit{Fonction}_2)}$

2.4 Discussion

Dans les sections précédentes, un formalisme permettant de représenter de manière unifiée les problèmes d'optimisation numérique a été présenté. On peut se demander quel apport l'utilisation d'un paradigme unique peut apporter. Dans la plupart des cas, les utilisateurs de solveurs numériques partent d'un outil existant et tentent de modéliser le problème à résoudre de sorte qu'il puisse être soluble par l'outil choisi. Cette approche est critiquable, car chaque problème possède une difficulté intrinsèque et l'on devrait au contraire pouvoir dans un premier temps modéliser son problème puis choisir le solveur adéquat une fois la modélisation correctement effectuée. De plus, le travail de recherche comporte, par nature, une grande part de prototypage rapide où pouvoir tester de nouvelles approches rapidement est important. Dans ce type de situation, utiliser le solveur le plus avancé de l'État de l'Art n'est pas toujours nécessaire et l'on peut éventuellement choisir à dessein un solveur "trop" puissant pour le problème à résoudre et sous-optimal en terme de temps de calcul. Inversement, il est courant qu'après avoir commencé un travail de recherche, on se rende compte qu'un solveur équivalent plus rapide ou présentant de meilleures caractéristiques existe. Dans ce cas, il est plus simple de l'intégrer à un formalisme existant plutôt que de devoir adapter son problème à un logiciel alors qu'algorithmiquement parlant il n'existe pas de différence notable. Pour terminer, il est arrivé que l'on commence tout d'abord par poser un problème robotique pour s'apercevoir qu'il serait bien mieux traité à l'aide d'un solveur dédié. Il est courant donc de devoir écrire un nouveau solveur, ou modifier un solveur existant, afin de pouvoir rendre le traitement du problème plus rapide. Dans ce cas également, un formalisme unique présente de nombreux avantages, notamment en terme de temps de conception.

Cette modélisation présente toutefois un certain nombre de limites qui peuvent être divisées en deux : les limites intrinsèques de la modélisation et celles qui ont été imposées par le langage de programmation choisi pour l'implémentation des algorithmes, à savoir le C++. Ces dernières seront abordées dans la section 2.5. Concernant les limites de la modélisation, différents éléments sont difficiles à modéliser.

La première limite se présente quand les variables d'optimisations appartiennent à des espaces possédant des structures différentes. Un exemple commun en robotique est le mouvement 2d sur un plan en translation et rotation. Trois variables caractérisent ce mouvement : x, y et θ. Les deux premières décrivent un point dans l'espace euclidien tandis que la dernière représente une rotation et appartient donc au groupe spécial orthogonal en deux dimensions $SO(2)$. La représentation des éléments de cet espace utilisant des réels est, par nature, discontinue à 2π. En choisissant un type unique pour représenter les variables d'optimisation, nous perdons la possibilité de réaliser un traitement spécifique selon l'espace mathématique auquel appartient la variable d'optimisation. Malheureusement, définir plusieurs types pour définir des éléments de différents groupes mathématiques est difficile. En effet, l'ensemble des variables d'optimisations n'est alors plus un tableau de réels, mais un n-uplet contenant des éléments des types différents. Il faut encore alors trouver un moyen de définir les opérations de groupe de manière efficace afin que le solveur puisse utiliser les opérations adéquates pour opérer sur le groupe tout en préservant ces caractéris-

tiques. Ce problème est bien plus visible lorsque l'on travaille sur des rotations en trois dimensions, $SO(3)$, pour lesquels de nombreuses représentations existent. Par exemple si l'on considérer les quaternions, on souhaite en général conserver un quaternion unitaire ce qui induit une contrainte supplémentaire qu'il faut pouvoir insérer dans le problème. La différenciation également suit des règles particulières qui représentent une branche des mathématiques appelée géométrie différentielle.

La seconde limite concerne la représentation des fonctions. En effet, le parti a été de se limiter à la représentation des fonctions de $[\iota] \to \iota$, soit pour les réels de $\mathbb{R}^n \to \mathbb{R}$. Ce choix de conception a été pour garder la taille des résultats calculés sous contrôle. En effet, pour une fonction quelconque, la représentation des dérivées partielles se fait sous la forme d'une matrice jacobienne, puis d'un tenseur d'ordre 3 pour les hessiens. Les tenseurs sont rarement utilisés par les solveurs, car ces objets de très grande taille sont lourds à manipuler. Qui plus est, il n'est pas toujours nécessaire d'évaluer toutes les valeurs des matrices ou tenseurs représentant les dérivées partielles d'une fonction et de ce fait, découper une fonction à valeurs vectorielles en plusieurs fonctions scalaires peut avoir du sens afin de limiter les calculs effectués.

La troisième limite est liée à la définition des types paramétrés. Il n'existe pas, a priori, de hiérarchie de classe entre deux types résultant d'un même type paramétré. Prenons par exemple le cas de l'optimisation non linéaire avec ou sans hessien : on a donc deux classes de problèmes reliés par une relation d'ordre qui semble naturelle : si l'on a la possibilité de calculer les hessiens, on devrait pouvoir le donner à la fois aux solveurs nécessitant les hessiens, mais également à ceux qui ne les utilise pas. La classe où l'évaluation des hessiens est possible est donc une classe plus "facile" que la classe où l'on n'a que des gradients. On peut, évidemment, transformer un problème avec hessien en problème sans hessien, mais ce n'est pas aussi simple que ce à quoi on pourrait s'attendre : soit τ un type paramétré et τ_1 τ_2 tel que $\tau_2 <: \tau_1$, on n'a pas $\tau_{\tau_2} <: \tau_{\tau_1}$. Autrement dit, les fonctions dérivent les unes des autres pour former une hiérarchie de classes interchangeables, mais pas les problèmes ! Pour changer de solveur, il faut donc instancier un type de problème différent et ensuite le passer au nouveau solveur. Cela représente généralement un travail court et facile, mais qui peut désarçonner les utilisateurs peu adeptes des langages de programmation utilisant des règles de typage complexe.

Face à la liste de ces critiques, on peut s'apercevoir qu'un élément manquant, car dépassant le cadre de ce travail, est l'absence de système de calcul formel sous-jacent. Développer un tel système tout en préservant un typage fort est une tâche ardue ; de ce fait, un compromis entre ergonomie et effort de modélisation a été trouvé pour permettre la réalisation de ce travail. Comme tout compromis, il handicape parfois l'expression de certains problèmes dont on aurait souhaité garder une formalisation plus proche de leurs natures mathématiques véritables.

2.5 RobOptim : une suite logicielle pour l'optimisation en robotique

Cette formalisation a été implémentée sous la forme d'une suite logicielle pour l'optimisation numérique en robotique. Le langage choisi pour l'implémen-

tation est le C++. Ce langage a la particularité d'être multiparadigme dont un très fort support à la fois de la programmation orientée objet, mais également de la métaprogrammation, c'est-à-dire la génération de familles de types via l'utilisation des "templates". Ce dernier point a été abondamment utilisé pour permettre de fournir une infrastructure de calcul sans impacter la performance des solveurs sous-jacents. En effet, il n'y a pas de coût à l'exécution induit par l'utilisation des templates [1] contrairement à l'utilisation de certaines techniques de programmation orientée objet comme les méthodes virtuelles. En effet, comme expliqué dans la section précédente, les méthodes virtuelles nécessitent une résolution et leur appel est donc plus coûteux qu'une fonction simple (Driesen et Hölzle, 1996).

RobOptim est une suite logicielle dont les composants peuvent être divisés en trois groupes :

le noyau fournissant les interfaces et les types nécessaires à l'implémentation de la formalisation décrite précédemment,

les solveurs implémentant les algorithmes de résolution pour un ou plusieurs,

les couches "métier" dédiées à la robotique et fournissant des fonctions régulièrement utilisées pour résoudre des problèmes d'optimisation. En particulier, le problème choisi pour montrer l'intérêt de cette approche concerne l'optimisation de trajectoire de marche pour un robot humanoïde ce qui a motivé le développement d'un ensemble d'outils pour l'optimisation de trajectoires.

2.5.1 RobOptim core et ses plug-ins

Le noyau de RobOptim fournit la modélisation mathématique des fonctions, la classe définissant les problèmes d'optimisation ainsi que les interfaces pour les solveurs. Le noyau, toutefois, ne contient aucun algorithme de résolution. Ce paquet principal contient également les outils mathématiques utiles pour résoudre les problèmes. Par exemple, on peut augmenter le niveau de différenciabilité d'une fonction via l'utilisation d'une méthode numérique se fondant sur la méthode des différences finies. Cette méthode permet d'évalue la dérivée d'une fonction en tirant des points aux alentours d'un point de référence pour évaluer la courbure de la fonction aux alentours de ce point. Plusieurs stratégies sont implémentées tirant deux ou cinq points selon les cas. Cette fonctionnalité est implémentée via l'utilisation du "Design Pattern" décorateur tel que décrit dans le livre de référence (Gamma et collab., 1994). Ce motif de conception a la particularité d'augmenter les capacités d'une classe sans en altérer le comportement propre. Dans ce cas, cette classe paramétrée prend un type de fonction en entrée, en hérite, tout en fournissant un niveau de calcul supplémentaire des dérivées. D'autres outils permettent de combiner des fonctions : les sommer, soustraire, multiplier entre elles ou bien encore les combiner. Un autre décorateur fournit également un cache afin d'éviter d'avoir à effectuer une nouvelle fois des calculs lourds qui ont déjà été demandés précédemment.

Au-delà de la définition des fonctions, le noyau fournit l'interface pour les solveurs. Deux solutions sont alors possibles : soit se lier à une ou des biblio-

1. À l'exception de quelques phénomènes impactant à la marge certains mécanismes de résolution dynamique. On pensera par exemple aux transtypages dynamiques ralentis par la prolifération des types engendrés par les mécanismes de métaprogrammation.

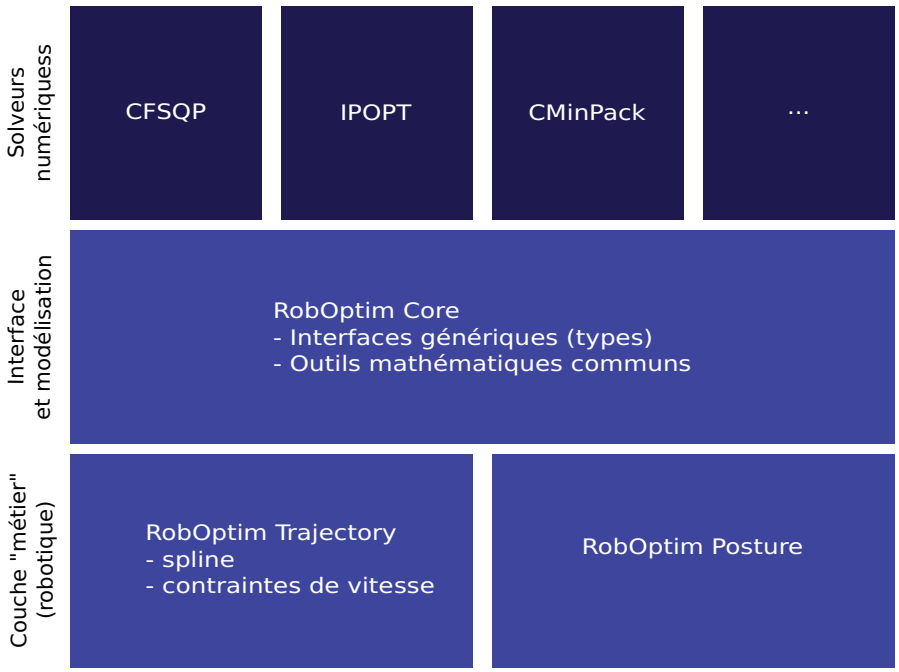

FIGURE 2.1 – Architecture logicielle de RobOptim

thèques tierces fournissant des algorithmes de résolution, soit utiliser le système de plug-in permettant d'en charger à l'exécution. L'objectif de ce système est de pouvoir complètement déléguer au système la résolution d'un problème : il y a un certain nombre de solveurs disponibles sur le système et l'on pourrait ainsi, avoir une sélection automatique de l'algorithme adéquat en fonction du type de problème en entrée. En pratique, ce mécanisme autorise un changement de solveur pendant l'exécution du programme.

2.5.2 Les solveurs

Il est important d'insister sur le fait que ce travail n'a pas pour but d'écrire un nouveau solveur ou bien de tenter d'apporter des améliorations à une ou des approches existantes. La littérature du domaine est foisonnante et bénéficie d'un effort de recherche important. Toutefois, l'objectif final de ce travail reste la possibilité d'utiliser dans un même paradigme plusieurs solveurs de types différents. Des plug-ins RobOptim ont été écrits pour trois solveurs :

CMinPack solveur permettant de résoudre des problèmes de type moindres carrés linéaires,

CFSQP solveur pour les problèmes non linéaires, contraintes égalités et inégalités, calcul du gradient requis (Craig et collab., 1997),

IPOPT solveur pour les problèmes non linéaires, contraintes égalités et inégalités, calcul du gradient et hessien requis [2].

Ces solveurs sont disponibles sous la forme de plug-in pour RobOptim core et peuvent être chargés dynamiquement dans n'importe quel problème d'optimisation RobOptim.

2.5.3 Optimisation de trajectoires avec RobOptim

Un paradigme de programmation unifié permet également d'unifier les outils de conception. En effet, l'écriture de bibliothèques de fonctions de coût et de contraintes prend tout son sens une fois que l'on a pu montrer que la même architecture est utilisable quelque soit la structure du problème à résoudre. De ce constat, RobOptim s'est enrichi d'une couche propre à la robotique, l'optimisation de trajectoires.

Cette couche définit des trajectoires robotiques sous forme de splines, c'est-à-dire de courbes paramétrées par des points de contrôle. Il est alors naturel de tenter de vouloir altérer ces points de contrôle afin de raffiner le comportement de la trajectoire : tenter de minimiser le temps nécessaire pour réaliser le mouvement, ou l'énergie ou tout autre critère adéquat.

Cette bibliothèque fournit des bornes sur les vitesses du système à la fois pour la vitesse de rotation et les vitesses linéaires. Le problème d'optimisation de trajectoires est un problème semi-infini dans la mesure où un souhait raisonnable est, par exemple, de borner la vitesse du robot à tout moment de la trajectoire. Malheureusement, la résolution de ce type de problèmes dans le cas général est difficile et la technique la plus couramment utilisée reste une discrétisation temporelle des contraintes. Ces stratégies présentent l'inconvénient d'augmenter très largement le nombre de contraintes du problème. De plus, le

2. Une version alternative qui ne nécessite pas le calcul du hessien est également fournie par le paquet logiciel.

pas de discrétisation est difficile à évaluer et a un très fort impact sur la vitesse de résolution. Enfin, si on ajoute la possibilité d'accélérer ou de décélérer la trajectoire au problème, le solveur peut tenter de changer la paramétrisation temporelle afin de placer toutes les contraintes au début du mouvement et pouvoir, par exemple, passer au travers d'un obstacle. Un tel comportement peut être évité en attachant la contrainte non pas à un moment temporel donné, mais plutôt à un point de la trajectoire indépendamment de sa paramétrisation temporelle. Il suffit pour cela de se donner une échelle fixe, de 0 à 1, représentant respectivement le début et la fin de la trajectoire, et permettant aux contraintes de se déplacer avec la reparamétrisation de la trajectoire. La discrétisation de contraintes utilisant ce type de représentation est supportée par RobOptim et permet de simplifier la résolution de problèmes d'optimisation de trajectoires.

2.6 Scénario d'utilisation : écriture d'une application robotique

Afin d'illustrer les propos des sections précédentes, nous allons désormais nous attacher à résoudre un problème de robotique avec le paradigme de RobOptim. L'exemple choisi est un problème de robotique humanoïde. Une stratégie possible pour pouvoir générer une trajectoire de marche est de tout d'abord générer un ensemble d'empreintes de pas et ensuite de générer un mouvement tel que le robot place successivement ses pieds dans les empreintes de pas générées tout en conservant son équilibre. Si cette stratégie est adoptée, le problème initial se ramène à trouver une pile de pas appropriée pour passer d'un point A à un point B. Ce dernier problème se ramène lui-même à trouver le mouvement sans collision dans le plan d'une boîte en rotation et translation. Une fois cette trajectoire trouvée, il suffit de placer les pas le long de la trajectoire de la boîte pour obtenir la pile de pas souhaitée.

Pour planifier le mouvement de la boîte, nous avons choisi d'utiliser un algorithme aléatoire de type RRT – Rapidly exploring Random Tree –. Cet arbre va tenter d'échantillonner l'espace $\mathbb{R}^2 \times SO(2)$ des translations et rotation en deux dimensions tout en validant que les configurations tirées ne violent pas de contrainte ainsi que le chemin amenant du nœud le plus proche au nouveau nœud tiré. Malheureusement, cette méthode rend des trajectoires rarement réalistes et qui ne sont pas toujours acceptables. Il est donc courant d'utiliser une méthode probabiliste pour trouver un premier chemin évitant les collisions qui est ensuite amélioré dans une deuxième passe réalisée par un algorithme d'optimisation. L'utilisation d'une méthode probabiliste permet d'éviter de "tomber" dans des minima locaux ce qui est un problème récurrent des méthodes d'optimisation. C'est la seconde passe, permettant de raffiner la trajectoire, que nous avons choisi d'exprimer dans le paradigme de RobOptim.

2.6.1 Définition de la trajectoire et fonction de coût

B-Splines et courbes paramétrées

Comme indiqué dans la section précédente, les trajectoires sont définies sous la forme d'une courbe paramétrée. Ce type de courbe appelé B-spline est défini comme une combinaison linéaire d'une famille de fonctions de base. Les deux

définitions ci-dessous permettent de calculer analytiquement ces courbes et sont tirées de (Malgouyres, 2002).

Définition 18. *Soit $n \in \mathbb{N}$, le nombre de points de contrôles. Soit $k \geq 1$, le degré des polynômes considérés. Soit $\Xi = (t_0, \ldots, t_{n+k-1})$ un vecteur appelé, vecteur nodal avec $t_0 \geq \ldots \geq t_{n+k-1}$. Les valeurs t_i sont appelées noeuds. Les fonctions de base $N_{i,k}$ que nous allons définir, et donc les courbes B-splines que nous construirons ensuite, sont polynomiales sur chaque intervalle $[t_i, t_{i+1}]$, pour $i = 0, \ldots, n-2$.*

La fonction $N_{i,k}$ est définie par récurrence, comme suit :
Pour $i = 0, \ldots, n+k-2$, on pose :

$$N_{i,1}(t) = \begin{cases} 1 & \text{si } t_i \leq t \leq t_{i+1} \text{ et } i = 0 \text{ ou } t \neq t_i \\ 0 & \text{sinon} \end{cases} \tag{2.27}$$

Pour $r \geq 2$ et pour $i = 0, \ldots, n+k-1-r$, on a la formule de récurrence :

$$N_{i,r}(t) = \frac{t - t_i}{t_{i+r-1} - t_i} N_{i,r-1}(t) + \frac{t_{i+r} - t}{t_{i+r} - t_{i+1}} N_{i+1,r-1}(t) \tag{2.28}$$

La formule permettant de calculer les $N_{i,r}$ à partir des $N_{j,r-1}$ s'appelle la formule de Cox-de-Boor. Lorsque les dénominateurs de la formule s'annulent, on pose $\frac{0}{0} = 0$. C'est le cas lorsque plusieurs nœuds t_i sont confondus.

Définition 19. *Soit maintenant $P_0, \ldots, P_{n-1}$ des points de contrôle. La courbe B-spline Q d'ordre n et de degré $k-1$ ayant Ξ pour vecteur nodal et les P_i pour points de contrôle s'exprime comme :*

$$\begin{cases} Q : [t_{k-1}, t_n] & \to & \mathbb{R}^3 \\ t & \mapsto & Q(t) = \sum_{i=0}^{n-1} P_i N_{i,k}(t) \end{cases} \tag{2.29}$$

La courbe Q s'exprime donc comme une somme des fonctions $N_{i,k}$, pondérées par les points de contrôle.

Dans notre application, nous avons choisi d'utiliser des B-splines uniformes de degré 3. Une B-spline est dit uniforme lorsque les intervalles temporels entre les points de contrôle définis par le vecteur nodal Ξ est constant entre chaque point de contrôle. Dans notre cas, cette valeur sera notée λ et fera partie des variables d'optimisation. Les autres variables d'optimisation sont les points de contrôle eux-mêmes, soit trois variables par point de contrôle : $(x, y) \in \mathbb{R}^2$ la position du robot dans le plan et $\theta \in S^1$ l'orientation du robot. Le vecteur de variables d'optimisation ainsi constitué est illustré par la Tableau 2.3.

λ	x_0	y_0	θ_0	$\ldots$	x_n	y_n	θ_n

TABLE 2.3 – Variables d'optimisation du problème.

Fonction de coût

La paramétrisation choisie permet d'accélérer la courbe. En effet, plus la valeur de λ décroît plus la courbe est accélérée. L'objectif est alors simplement de tenter de réaliser le mouvement le plus vite possible. On réalise alors une optimisation temps minimal utilisant la fonction de coût suivante :

$$\text{Coût}(\mathbf{x}) \quad = \quad (1, 0, \ldots, 0)^T \mathbf{x} \tag{2.30}$$

Dans les faits, le solveur va progressivement saturer les contraintes en accélérant la courbe. En particulier jusqu'à ce que la contrainte de vitesse détaillée ci-dessous, soit saturée. La Figure 2.2 montre le processus de saturation d'une contrainte sur la vitesse dans le cadre d'une spline unidimensionnelle.

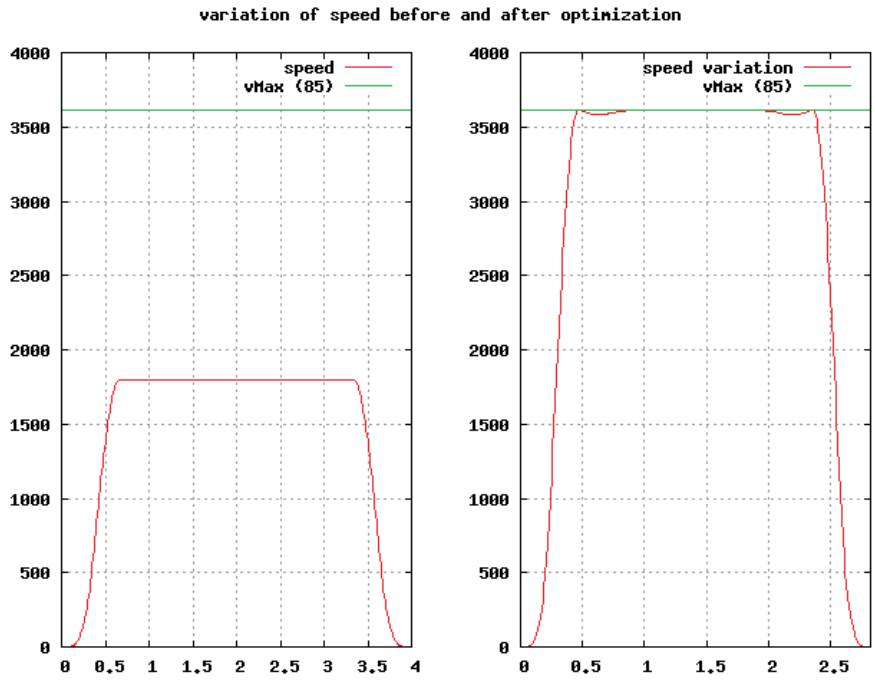

FIGURE 2.2 – Saturation de la contrainte de vitesse.

2.6.2 Contraintes

Contrainte sur la vitesse

Le but de la contrainte sur les vitesses est double : s'assurer de la correction de la trajectoire générée tout en privilégiant la marche en avant ou en arrière par rapport à la marche sur le côté. Outre l'aspect anthropomorphe, les robots humanoïdes ont un comportement dégradé quand ils marchent "en crabe". L'espace accessible par les jambes est réduit dans cette direction et la forme des semelles du robot a tendance à le faire glisser dans ce cas. Les contraintes sur

les vitesses sont dupliquées afin de s'appliquer à la fois au pied gauche et au pied droit. La formulation de la contrainte pour tous les points de la trajectoire t et pour chaque pied est :

$$\begin{aligned} &\forall t, \forall \text{pied} \in \{\text{gauche}, \text{droite}\}, \\ &\text{ContrainteVitesse}_{(t,\text{pied})}(\mathbf{x}) = (\frac{v^x_{\text{pied}}}{v^x_{\max}})^2 + (\frac{v^y_{\text{pied}}}{v^y_{\max}})^2 - 1 \end{aligned} \tag{2.31}$$

v^x_{pied} et v^y_{pied} étant respectivement les vitesses frontales et orthogonales du pied considéré. $v^x_{\max}$ et $v^y_{\max}$ respectivement les vitesses frontales et orthogonales maximums du robot. Plus le ratio $\frac{v^x_{\max}}{v^y_{\max}}$ est élevé, plus le robot privilégiera la marche en avant pendant l'optimisation au détriment de la vitesse d'exécution du mouvement.

Cette contrainte impose à la vitesse des pieds de rester dans une ellipse allongée vers l'avant. Cette contrainte impose un mouvement privilégiant la marche en avant et s'assure que le mouvement des pieds est suffisamment lent pour pouvoir être jouée sur le robot. Afin d'être certain que la contrainte est vérifiée en permanence, le temps est discrétisé et on ajoute la contrainte pour tous les pas de temps.

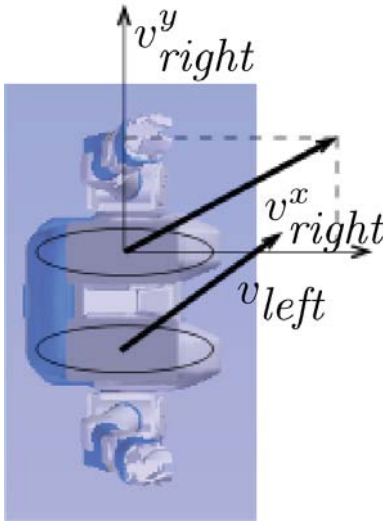

FIGURE 2.3 – Vitesses séparées du pied gauche et du pied droit.

Contrainte sur les obstacles

Les contraintes sur les obstacles se définissent naturellement : on souhaite maintenir une distance minimale entre le robot et chaque obstacle à tout moment. Comme avec la contrainte précédente, on discrétise le temps afin de vérifier régulièrement la contrainte le long de la trajectoire. La contrainte en tous points de la trajectoire t et pour tous les obstacles est la suivante :

Scenarii	Planification	Optimisation	*Calcul complet*
Passage entre deux chaises	8.58s	47.22s	*2 min 05.55 s*
Salle réaliste avec de nombreux obstacles	4.14 s	1 min 9.35 s	*4 min 5.11 s*

TABLE 2.4 – Temps de calculs. Le temps de calcul complet comprend la planification, l'optimisation et la phase de génération de mouvement corps complet.

$$\forall t, \forall j, \text{Distance}(\mathcal{R}(t), \mathcal{O}_j) \geq d_{\min} \tag{2.32}$$

$\mathcal{R}$ représente ici la position du robot et $\mathcal{O}_j$ la position de l'obstacle j de l'environnement. d_{min} représente une distance de sécurité déterminée empiriquement entre les obstacles et le robot.

2.6.3 Résolution et résultats expérimentaux

La fonction de coût est une fonction linéaire, mais les contraintes sont non linéaires. Il est nécessaire de définir des types RobOptim pour ces fonctions, le type de problème associé : $\text{Problème}_{\text{FonctionLinéaire},\text{Fonction}_1}$. Le gradient analytique a été déterminé pour toutes les fonctions afin d'accélérer les calculs. La Figure 2.4 montre un exemple de chemin optimisé par le solveur CFSQP en simulation et la Figure 2.5 un exemple d'application sur le robot. Le Tableau 2.4 illustre les temps de calcul rencontrés sur deux scénarios.

2.7 Conclusion

Ce chapitre a été dédié à la conception d'une modélisation orientée objet pour la résolution de problèmes d'optimisation numérique ainsi qu'à son implémentation, la suite logicielle RobOptim. Après avoir observé les solveurs disponibles sur le marché, on peut facilement s'apercevoir de l'écart considérable entre d'un côté la théorie mathématique et son implémentation en pratique. En pratique, chaque solveur vient avec son formalisme, ses interfaces et force ses utilisateurs à apprendre une nouvelle formalisation pour chaque nouveau solveur utilisé. De ce fait, réaliser des comparaisons est malaisé, l'apprentissage inutilement difficile et les interfaces souvent peu claires ou mal documentées. En fournissant un point d'entrée pour plusieurs algorithmes, il n'y a qu'une interface et un paradigme à expliquer ce qui présente un gain net vis-à-vis de l'approche habituelle. De plus, de nombreux outils sont utiles en optimisation tout en étant décorrélés de l'algorithme de résolution. Le calcul des gradients par différences finies est un bon exemple. Dans RobOptim, il a été intégré à la fois pour pouvoir éviter d'avoir à calculer les gradients de manière analytique, mais il peut également servir à vérifier le gradient analytique. Du point de vue du logiciel, l'accent a été mis sur la création d'un ensemble de bibliothèques logicielles simples à utiliser et assurant la correction du problème posé. Cette correction passe par l'utilisation d'un système de typage fort permettant de détecter à la compilation les problèmes qui

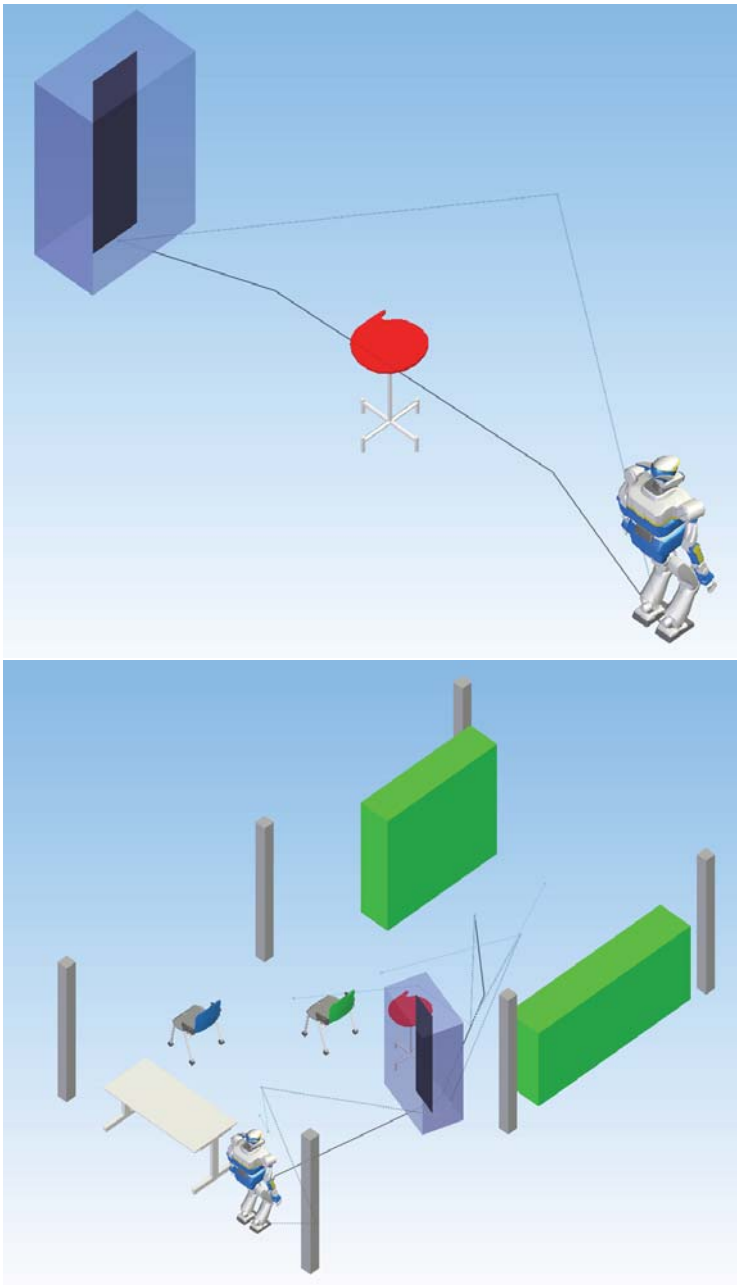

FIGURE 2.4 – Exemples d'optimisation d'une trajectoire en présence d'obstacles.

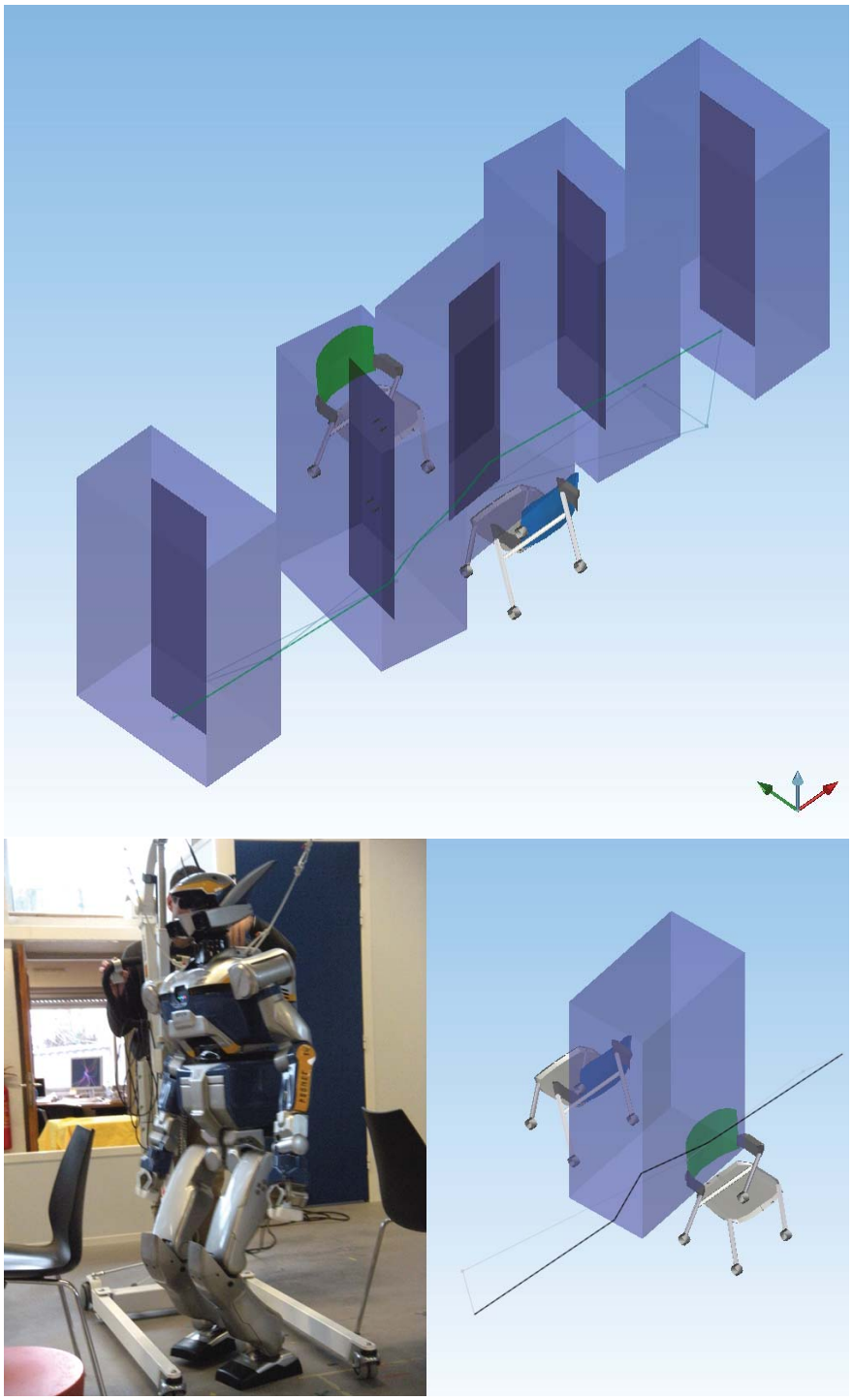

FIGURE 2.5 – Résultat expérimental sur le robot humanoïde HRP-2.

peuvent se poser. En effet, RobOptim interdit, à la compilation, la construction d'un problème invalide : si certains gradients sont manquants ou bien si certains types de fonction sont incorrects, la compilation du fichier échouera, car les contraintes de typage ne seront pas respectées. Malgré ce système de typage imposant des règles strictes, une grande flexibilité a été gardée dans le choix de l'algorithme qui peut s'effectuer à l'exécution. Le système de plug-in de RobOptim permettant de charger les algorithmes dynamiquement pendant l'exécution du programme. La suite logicielle a été validée sur un exemple réel de robotique qu'est l'optimisation de trajectoire de marche pour un robot humanoïde.

3 Suivi de trajectoire

I can calculate the motion of heavenly bodies, but not the madness of people.

Isaac Newton

Ce chapitre est dédié au problème de la génération et de l'exécution asservie de trajectoires pour un robot humanoïde. Les robots humanoïdes sont des robots ayant une forme anthropomorphe : une tête, deux bras, deux jambes et un torse. La locomotion est donc bipède et les robots humanoïdes ont la possibilité d'effectuer des tâches dextres. De ces choix résulte un système hautement dimensionné pour lequel il est difficile de générer des trajectoires et dont le contrôle est délicat, l'équilibre d'un système bipède étant précaire par nature. Au long de ce chapitre, nous verrons comment modéliser ce système, comment représenter et calculer des mouvements pour ce dernier afin qu'il puisse se mouvoir et réaliser différentes tâches. Seront ensuite étudiées les différentes approches de la littérature ainsi que leurs forces et leurs faiblesses. La troisième section portera sur l'apport de cette thèse au problème de l'exécution de trajectoire via la proposition d'une architecture de contrôle pour les robots humanoïdes asservie capteur. Les résultats expérimentaux seront ensuite détaillés pour illustrer une tâche dans laquelle le système proposé se révèle utile. Enfin, les prospectives montreront quelles sont les possibilités d'améliorations de l'approche proposée et quels futurs travaux restent à réaliser.

3.1 Génération de mouvements

3.1.1 Structure robotique

Un robot est un système composé par des actionneurs – moteurs –, des capteurs et des unités de calcul logique – ordinateurs – travaillant de concert pour constituer une entité cohérente pouvant impacter le monde réel afin de réaliser l'objectif qui lui a été confié. En ce qui concerne la génération de mouvements, deux éléments sont primordiaux : d'une part les capacités motrices du système, modélisées sous la forme d'un arbre cinématique et d'autre part sa forme dans le monde représenté par la forme géométrique des différentes parties du système. La forme géométrique d'un robot variant généralement sous l'action de ses actionneurs, elle est donc conditionnée par l'état de l'arbre cinématique.

Définition 20. *Un arbre cinématique est un ensemble d'articulations formant un arbre. Soit $\mathcal{A}$ un arbre cinématique, on a :*

$$\mathcal{A} = \emptyset \cup \{(\mathcal{J}_0, \mathcal{C}_0), \ldots, (\mathcal{J}_n, \mathcal{C}_n)\} \tag{3.1}$$

Un arbre est donc, soit l'ensemble vide, soit une liste de joints associés à un sous-arbre. Cette définition permet de construire récursivement un arbre cinématique.

À cet arbre de joints correspond une liste de joints, les joints sont habituellement référencés par leur ordre dans cette liste sous la notation $\mathcal{J}_n$ où n est la position du joint dans la liste.

Définition 21. *Le joint $\mathcal{J}_n$ modélise le mouvement du repère $n+1$ par rapport au repère n selon un certain nombre de paramètres appelés degrés de liberté. Chaque joint définit implicitement son repère attaché. Le repère attaché à $\mathcal{J}_0$ est habituellement appelé repère "monde" et est noté $\mathcal{W}$.*

Un joint peut donc se modéliser comme une fonction :

$$\left\{ \begin{array}{rccc} \mathcal{J}_n : & \mathbb{R}^m & \rightarrow & SE(3) \\ & \mathbf{q} & \mapsto & \mathcal{J}_n(t) \end{array} \right. \tag{3.2}$$

Dans l'équation le joint $\mathcal{J}_n$ possède m degrés de liberté permettant de déterminer la transformation relative entre le repère n et $n+1$ qui est un élément du groupe spécial euclidien de dimension 3. Ces fonctions permettent donc d'exprimer des contraintes sur les mouvements du corps du robot au niveau des articulations actionnées.

Plusieurs types de joints sont habituellement utilisés en robotique, mais nous n'allons ici en détailler que deux :

Joint libre – ou joint flottant –. Ce joint comporte six degrés de liberté, soit le nombre de paramètres nécessaire et suffisant pour spécifier une pose dans $SE(3)$. Autrement dit, ce joint n'impose aucune contrainte particulière sur le mouvement et correspond donc à la fonction identité, à la représentation de la pose prête.

Joint rotation. Ce joint comporte un seul degré de liberté. Dans ce cas, la position du prochain corps du robot est définie par la valeur de ce degré de liberté, interprété comme une rotation d'un angle équivalent autour d'un axe. A chaque joint rotation est donc, soit associé un axe de rotation, soit un axe standard est choisi : par exemple, toutes les rotations seront réalisées autour de l'axe X.

Une fois cette structure construite, une attente naturelle est pouvoir exprimer son état. C'est l'objectif du vecteur de configuration.

Définition 22. *Soit $\mathbf{q}$ le vecteur de configuration de l'arbre cinématique $\mathcal{C}$. Soit $\mathcal{J}_0, \ldots, \mathcal{J}_n$ l'ensemble des articulations de $\mathcal{C}$. Le vecteur de configuration a pour dimension : $\sum_{i=0}^{n} \#\mathcal{J}_i$, la somme des arités des articulations interprétées comme fonctions, c'est-à-dire le nombre de degrés de liberté de toutes les articulations.*

Les valeurs du vecteur de configuration sont alors les valeurs des degrés de liberté des joints concaténés les uns aux autres selon un parcours de l'arbre donné. En général, le parcours main gauche de l'arbre est utilisé.

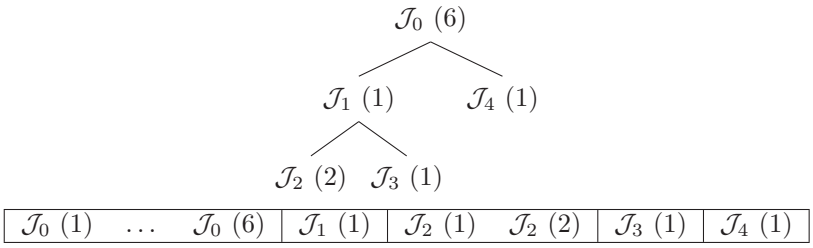

FIGURE 3.1 – Un exemple de chaîne cinématique et son vecteur de configuration correspondant. Les nombres entre parenthèses indiquent le nombre de degrés de liberté dans l'arbre ou le numéro du degré de liberté dans le joint pour le vecteur de configuration.

On notera que l'arbre cinématique d'un robot humanoïde est souvent composé d'un joint libre à la racine suivie de joints rotation pour le reste de l'arbre. À partir de maintenant, cette forme sera systématiquement utilisée ce qui nous permettra de diviser le vecteur de configuration en deux parties : d'un côté la position du robot dans le plan, définie par une sous-partie des degrés de liberté du joint libre à la racine, le repère de ce joint définissant de ce fait le repère monde $\mathcal{W}$ et d'autre part les degrés de liberté internes correspondant à une réalité mécanique. Sous l'hypothèse que le robot se déplace sur un sol plan, on aura donc :

$$\begin{aligned} \mathbf{x} &= [x, y, r_z] \\ \mathbf{q} &= (\mathbf{x}, [z, r_x, r_y, \mathbf{q}_{\text{int}}]) \in \mathcal{C} = \text{SE}(2) \times \mathcal{C}_{\text{int}} \end{aligned} \tag{3.3}$$

Les six paramètres du joint racine sont : $[x, y, z, r_x, r_y, r_z]$ et définissent une position dans l'espace 3D. Toutefois, dans la mesure où la position du robot est contrainte par la physique, c'est-à-dire que l'on peut raisonnablement supposer un contact planaire entre les pieds du robot et le sol, on peut se contenter de considérer la position du robot en deux dimensions dans le plan. Pour cette raison, on peut considérer : $\mathbf{x} = [x, y, r_z] \in \text{SE}(2)$ pour déterminer la position du robot dans l'espace. $\mathcal{C}$ représente alors l'espace des configurations et $\mathcal{C}_{\text{int}}$ est l'espace des configurations correspondant aux degrés de liberté internes uniquement. Ce découpage peut sembler arbitraire au premier abord, mais prend son sens quand on considère quels degrés de liberté peuvent dériver suite à des erreurs de modélisation ou d'exécution : la position du robot dans le plan peut se révéler fausse et doit être estimée et corrigée constamment alors que les degrés de liberté $[z, r_x, r_y]$ ne peuvent pas subir de dérives durant le mouvement. Le problème de la correction des erreurs est détaillé plus loin dans ce chapitre et le problème de la localisation d'un robot humanoïde est abordé dans le chapitre suivant.

Une fois cette chaîne cinématique définie, on peut y attacher les informations concernant les corps du robot, leur réalité physique : forme et information inertielles en particulier. La forme est stockée sous la forme d'une soupe de polygones réalisant une approximation de la forme originale du corps du robot. Quant aux informations inertielles, elles consistent en le poids du corps, la position de son centre de masse et sa matrice d'inertie.

3.1.2 Cinématique directe et inverse

Cinématique directe

À partir de la structure définie dans la section précédente, la première opération que l'on souhaite réaliser est de pouvoir déterminer la position des corps du robot en fonction de la configuration de ce dernier. Cette opération, appelée calcul de la géométrie directe, consiste en la fonction suivante :

$$\text{GéométrieDirecte}_{\mathcal{C}}(q) = \{{}^{\mathcal{W}}\text{M}_{\mathcal{J}_0}, \ldots, {}^{\mathcal{W}}\text{M}_{\mathcal{J}_n}\} \tag{3.4}$$

La notation ${}^{\mathcal{A}}\text{M}_{\mathcal{B}}$ représente la transformation qui donne la position du repère $\mathcal{B}$ relative au repère $\mathcal{A}$. L'intérêt de cette représentation est de pouvoir assembler les transformations facilement, en effet, on a :

$$^{\mathcal{A}}\mathrm{M}_{\mathcal{C}} = {}^{\mathcal{A}}\mathrm{M}_{\mathcal{B}}{}^{\mathcal{B}}\mathrm{M}_{\mathcal{C}} \tag{3.5}$$

Les transformations géométriques sont des éléments du groupe spécial euclidien SE(3). Le produit utilisé ici est donc le produit de groupe. Dans le cas où ces éléments sont représentés sous la forme de matrices homogènes, le produit de groupe est le produit matriciel usuel.

La géométrie directe donne donc la position de tous les corps dans le repère monde. Dans la section précédente, on a défini un joint comme la position du repère attaché au joint suivant par rapport à la position du joint actuel.

Définition 23. *Soit $\mathcal{J}_n$ un joint de l'arbre cinématique et $\Delta = \{\mathcal{J}_0, \ldots, \mathcal{J}_n\}$ le chemin dans l'arbre de la racine au noeud $\mathcal{J}_n$. Soit $\mathbf{q}_i$ les degrés de liberté du joint $\mathcal{J}_i$. La position du joint $\mathcal{J}_n$ dans le repère monde est :*

$$^{\mathcal{W}}M_{\mathcal{F}_i} = \prod_{j \in \Delta} \mathcal{J}_j(\mathbf{q}_j) \tag{3.6}$$

Cinématique inverse

La Définition 23 permet de construire la position de tous les corps du robot dans le repère monde attaché au joint racine de la structure. Cependant, lors de la conception d'un problème robotique, il est courant de vouloir, par exemple, positionner un effecteur du robot – un préhenseur par exemple – à un endroit particulier de l'espace euclidien. Dans ce cas, le problème inverse doit être résolu : trouver une configuration telle qu'un corps particulier du robot atteigne une position donnée. Ce problème est appelé géométrie inverse.

Une difficulté de ce problème est que l'on doit inverser une équation dont le nombre de variables correspond au nombre de degrés de liberté du robot et dépend évidemment de sa structure. Si l'on souhaite positionner un corps à un endroit précis en rotation et translation, six variables sont nécessaires. De ce fait, selon le nombre de degrés de liberté du système et la position à atteindre, aucune solution, un nombre fini ou infini de solutions peuvent exister. La résolution analytique n'étant pas toujours possible, les outils numériques ont montré leur capacité à résoudre efficacement ce problème.

3.1.3 Mouvements dynamiques d'un corps dans l'espace

La cinématique directe permet de calculer la vitesse de corps dans l'espace indépendamment de leur réalité physique. Cependant, ce n'est pas suffisant pour pouvoir générer un mouvement complexe : le poids des corps du robot, leurs propriétés inertielles ainsi que les forces exercées s'appliquant au robot doivent être prises en compte pour pouvoir donner un mouvement qui n'est pas seulement viable dans un monde sans physique, mais également dans le monde réel où les lois de la physique s'appliquent.

Pour ce faire, nous allons commencer par rappeler les notions de mécanique nécessaires à la génération de mouvements et les différents modèles simplifiés utilisés en robotique humanoïde. Tout l'enjeu de la robotique humanoïde se joue ici : d'un côté on ne peut ignorer la physique pour générer un mouvement de marche, mais de l'autre le coût important de la résolution des modèles de

la physique classique newtonienne empêche l'écriture de contrôleurs suffisamment rapides pour pouvoir fonctionner en temps réel sur un robot. Il y a ici un compromis à trouver qui fait toute la richesse de la robotique humanoïde : quelles simplifications adopter ? quels calculs peuvent être réalisés à l'avance, une fois pour toutes, et quels calculs doivent être évalués en ligne ? Nous allons tout d'abord nous attacher à rappeler les lois de la mécanique classique avant de nous intéresser aux différentes simplifications qui permettent de rendre le problème tractable.

Quelques éléments de mécanique

Lois fondamentales de la mécanique

Définition 24. *(Relation fondamentale de la dynamique)*
Tout point de masse m de position M soumis à un ensemble de forces dont la somme est $\boldsymbol{F}$ est mû par un mouvement d'accélération γ donnée par la relation suivante :

$$\boldsymbol{F} = m\ddot{\boldsymbol{x}} \tag{3.7}$$

Définition 25. *(Principe d'action-réaction)*
Soient P_1 et P_2 deux points de masses respectives m_1 et m_2. Si $\boldsymbol{F}_{1/2}$ est la force exercée par P_1 sur P_2 et $\boldsymbol{F}_{2/1}$ la force exercée par P_2 sur P_1, alors :

$$\boldsymbol{F}_{1/2} = -\boldsymbol{F}_{2/1} \tag{3.8}$$

Définition 26. *(Principe de colinéarité)*
Soient P_1 et P_2 deux points de masses respectives m_1 et m_2. La force $\boldsymbol{F}_{1/2}$ exercée par P_1 sur P_2 est colinéaire au vecteur $\vec{P_1P_2}$

Systèmes de points

Définition 27. *(Moment d'un vecteur en un point)*
Soit $\boldsymbol{V}$ un vecteur de point d'application M. Le moment en un point O de $\boldsymbol{V}$ est défini par le produit vectoriel suivant :

$$\mathcal{M}_O^{\boldsymbol{V}} = \vec{OM} \times \boldsymbol{V}$$

Dans toute cette section, $(P_i)_{i=1,\dots,n}$ est un système de points de masses respectives m_i, de positions respectives M_i, de vitesses respectives $\mathbf{v}_i$ et d'accélérations respectives γ_i.

Pour chaque point P_i, on peut écrire la relation fondamentale de la dynamique (3.7) en distinguant les forces internes $\mathbf{F}_{j/i}$ appliquées par les autres points du système et les forces externes $\mathbf{F}_i^{ext}$ issues de causes externes au système :

$$\sum_{j=1, j\neq i}^{n} \mathbf{F}_{j/i} + \mathbf{F}_i^{ext} = m_i\gamma_i$$

On peut également écrire le moment de chaque membre de cette égalité en un point O :

$$\sum_{j=1, j\neq i}^{n} \vec{OM_i} \times \mathbf{F}_{j/i} + \vec{OM_i} \times \mathbf{F}_i^{ext} = m_i \vec{OM_i} \times \gamma_i$$

En sommant les deux égalités précédentes pour tous les points du système, les termes relatifs aux forces internes s'annulent par les principes d'action-réaction et de colinéarité. En effet, si on considère deux points P_i et P_j du système les forces relatives à ces deux points vérifient :

$$\mathbf{F}_{i/j} + \mathbf{F}_{j/i} = 0$$

et

$$\begin{aligned} O\vec{M}_i \times \mathbf{F}_{j/i} + O\vec{M}_j \times \mathbf{F}_{i/j} &= O\vec{M}_i \times \mathbf{F}_{j/i} - O\vec{M}_j \times \mathbf{F}_{j/i} \\ &= M_j\vec{M}_i \times \mathbf{F}_{j/i} \\ &= 0 \end{aligned}$$

La dernière égalité découle du principe de colinéarité. Il en résulte les égalités suivantes :

$$\sum_{i=1}^{n} \mathbf{F}_i^{ext} = \sum_{i=1}^{n} m_i \gamma_i \tag{3.9}$$

$$\sum_{i=1}^{n} O\vec{M}_i \times \mathbf{F}_i^{ext} = \sum_{i=1}^{n} m_i O\vec{M}_i \times \gamma_i \tag{3.10}$$

Définition 28. *(Centre de masse)*
Soit G le centre de masse et M la masse totale du système de points :

$$G = \frac{1}{M} \sum_{i=1}^{n} m_i M_i$$

En dérivant deux fois cette égalité, on en déduit l'accélération du centre de masse du système :

$$\gamma_G = \frac{1}{M} \sum_{i=1}^{n} m_i \gamma_i$$

Cette expression injectée dans (3.9) nous donne la propriété suivante bien connue : la somme des forces appliquées à un système est égale à la masse du système que multiplie l'accélération de son centre de masse.

$$\sum_{i=1}^{n} \mathbf{F}_i^{ext} = M\gamma_G$$

Cette propriété découle de la relation fondamentale de la dynamique et du principe d'action-réaction.

Torseurs

Définition 29. *(Torseur)*
Un torseur est un champ de vecteurs $\boldsymbol{M}$ qui vérifie la propriété suivante : il existe un vecteur $\boldsymbol{R}$ tel que pour tout couple de points O et P la relation suivante soit satisfaite :

$$\boldsymbol{M}_O = \boldsymbol{M}_P + \vec{OP} \times \boldsymbol{R}$$

$\boldsymbol{R}$ est unique. Il est appelé la résultante *du torseur noté :*

$$\{\boldsymbol{M}_O | \boldsymbol{R}\}_O$$

Exemple 5. *(Le torseur des vitesses d'un solide)*
Le champ de vecteurs des vitesses d'un solide est un torseur dont la résultante est le vecteur vitesse de rotation généralement noté $\vec{\omega}$.

Exemple 6. *(Le champ des moments des forces extérieures sur un système)*
Si l'on considère le système de points P_i soumis à des forces extérieures $\boldsymbol{F}_i^{ext}$, on définit le moment de ces forces en un point O par :

$$\mathcal{M}_O = \sum_{i=1}^{n} O\vec{M}_i \times \boldsymbol{F}_i^{ext}$$

On vérifie alors aisément que ce champ de moment est un torseur dont la résultante est la somme des $\boldsymbol{F}_i^{ext}$. En effet,

$$\begin{aligned}
\mathcal{M}_O &= \sum_{i=1}^{n} O\vec{M}_i \times \boldsymbol{F}_i^{ext} = \sum_{i=1}^{n} (\vec{OP} + P\vec{M}_i) \times \boldsymbol{F}_i^{ext} \\
&= \sum_{i=1}^{n} \vec{OP} \times \boldsymbol{F}_i^{ext} + P\vec{M}_i \times \boldsymbol{F}_i^{ext} \\
&= \mathcal{M}_P + \vec{OP} \times \sum_{i=1}^{n} \boldsymbol{F}_i^{ext}
\end{aligned}$$

Torseur cinétique Le torseur cinétique d'un système de points est le champ, noté $\vec{\sigma}$, des moments de quantités de mouvement du système :

$$\vec{\sigma}_O = \sum_{i=1}^{n} O\vec{M}_i \times m_i \mathbf{v}_i$$

La résultante du torseur cinétique est la quantité de mouvement du système :

$$\mathbf{P} = \sum_{i=1}^{n} m_i \mathbf{v}_i$$

Si O est un point fixe, on peut dériver la relation suivante pour obtenir :

$$\begin{aligned}
\dot{\vec{\sigma}}_O &= \sum_{i=1}^{n} \mathbf{v}_i \times m_i \mathbf{v}_i + O\vec{M}_i \times m_i \gamma_i \\
&= \sum_{i=1}^{n} O\vec{M}_i \times m_i \gamma_i
\end{aligned}$$

Cette dernière égalité nous permet d'énoncer la propriété suivante :

Propriété 1. *la dérivée du moment cinétique d'un système de points par rapport à un point fixe est égale au moment des forces extérieures par rapport à ce point sur le système :*

$$\dot{\vec{\sigma}}_0 = \mathcal{M}_O^{\boldsymbol{F}^{ext}} \tag{3.11}$$

La dérivée de la résultante cinétique est égale à la résultante des forces extérieures :

$$\dot{\boldsymbol{P}} = \sum_{i=1}^{n} \boldsymbol{F}_i^{ext} \tag{3.12}$$

Remarque : il est aisément vérifiable que la propriété du moment cinétique est valide également au centre de masse même si celui-ci n'est pas fixe :

$$\dot{\vec{\sigma}}_G = \mathcal{M}_G^{\mathbf{F}^{ext}}$$

Cas des solides Un solide est un ensemble d'atomes considérés comme des masses ponctuelles. Les efforts que ces points exercent les uns sur les autres respectent les principes d'action-réaction et de colinéarité. Une approximation courante consiste à considérer les solides comme des milieux continus. Cela a pour conséquence de remplacer les sommes par des intégrales dans les résultats précédents.

Définition 30. *(Torseur cinétique)*

Dans le cas d'un solide, la résultante cinétique ou quantité de mouvement devient :

$$P = \int_S \boldsymbol{v}_M dm(M)$$

où $dm(M)$ est la mesure de densité massique et $\boldsymbol{v}$ le champ des vitesses qui, rappelons-le, est un torseur :

$$\boldsymbol{v}_M = \boldsymbol{v}_G + \vec{MG} \times \vec{\omega}$$

Il en résulte que :

$$\begin{aligned} P &= \int_S (\boldsymbol{v}_G + \vec{MG} \times \vec{\omega}) dm(M) \\ &= M\boldsymbol{v}_G + \int_S (\vec{MG}) dm(M) \times \vec{\omega} \\ &= M\boldsymbol{v}_G \end{aligned}$$

par définition du centre de masse.

Le moment cinétique au centre de masse s'écrit :

$$\begin{aligned} \vec{\sigma}_G &= \int_S G\vec{M} \times \mathbf{v}_M dm(M) \\ &= \int_S G\vec{M} \times (\mathbf{v}_G + \vec{\omega} \times G\vec{M}) dm(M) \\ &= \int_S G\vec{M} dm(M) \times \mathbf{v}_G + \int_S G\vec{M} \times (\vec{\omega} \times G\vec{M}) dm(M) \\ &= \int_S G\vec{M} \times (\vec{\omega} \times G\vec{M}) dm(M) \end{aligned}$$

par définition du centre de masse.

Remarque : Il est intéressant de remarquer les deux points suivants :

1. L'opérateur $\hat{J}$ qui à $\vec{\omega}$ associe $\vec{\sigma}_G$ est linéaire. Sa matrice dans toute base orthonormée est symétrique positive définie.
2. $\vec{\omega}$ et $\vec{\sigma}_G$ ne sont en général pas colinéaires.

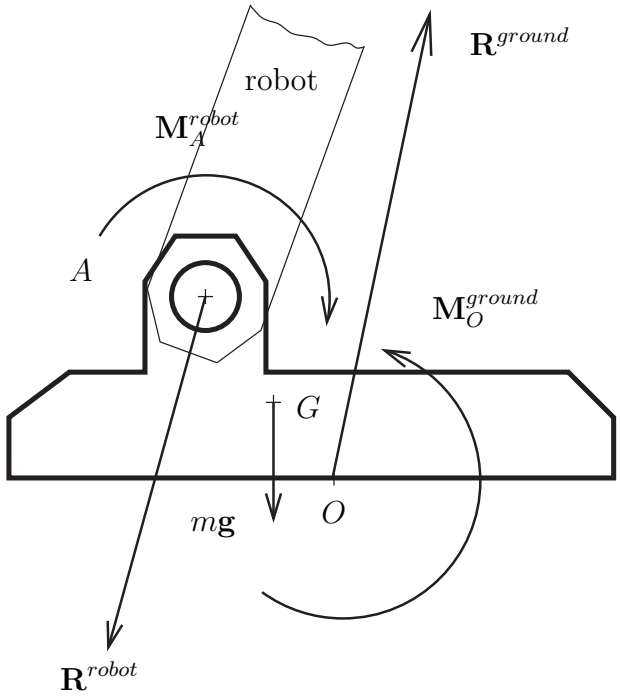

FIGURE 3.2 – L'équilibre statique du pied implique que le torseur des efforts extérieurs : l'effet de la gravité, la réaction du sol et l'action du robot sur le pied s'annulent.

Zero-Momentum Point (ZMP)

Nous considérons maintenant un robot humanoïde muni de deux pieds plats en phase de simple contact. La définition et l'analyse du ZMP reposent sur la condition de stabilité du pied du robot en contact avec le sol supposé horizontal.

Considérons donc le pied en contact avec le sol. Les actions extérieures appliquées sur ce pied sont (Figure 3.2) :

1. l'action du robot privé de son pied sur la cheville, représentée par le torseur :
$$\left\{\mathbf{M}_A^{robot}|\mathbf{R}^{robot}\right\}_A$$
où A est le centre de l'articulation de la cheville.
2. l'action de la gravité sur le pied représentée par le torseur
$$\{\mathbf{0}|m\mathbf{g}\}_G$$
où G est le centre de masse du pied, $\mathbf{g}$ le vecteur d'accélération due à la gravité et m la masse du pied.
3. l'action du sol sur le pied représentée par le torseur
$$\left\{\mathbf{M}_O^{ground}|\mathbf{R}^{ground}\right\}_G$$
où O est l'origine du repère supposée appartenir au plan du sol.

L'hypothèse de stabilité du pied sur le sol s'exprime donc par 6 équations qui lient les résultantes et moments ci-dessus. En supposant connus le poids et le centre de masse du pied, l'action du robot sur le pied, on peut en déduire l'action du sol sur le pied puis vérifier que cette action est possible étant donné que les efforts de contact du sol sur le pied ont une composante verticale positive. Ecrivons ces équations :

$$\begin{aligned}\mathbf{R}^{robot} + m\mathbf{g} + \mathbf{R}^{ground} &= 0\\ \mathbf{M}_A^{robot} + \vec{OA} \times \mathbf{R}^{robot} + \vec{OG} \times m\mathbf{g} + \mathbf{M}_O^{ground} &= 0\end{aligned}$$

Ainsi l'action du sol sur le pied est donnée par :

$$\begin{aligned}\mathbf{R}^{ground} &= -\mathbf{R}^{robot} - m\mathbf{g}\\ \mathbf{M}_O^{ground} &= -\mathbf{M}_A^{robot} - \vec{OA} \times \mathbf{R}^{robot} - \vec{OG} \times m\mathbf{g}\end{aligned}$$

Définition du ZMP Le ZMP est le point du sol en lequel les composantes horizontales du moment du sol sur le pied sont nulles. Nous le notons Z. Exprimons le moment des actions du sol sur le pied en $Z = (Z_x, Z_y, 0)$:

$$\begin{aligned}\mathbf{M}_Z^{ground} &= \mathbf{M}_O^{ground} + \vec{ZO} \times \mathbf{R}^{ground}\\ &= \begin{pmatrix} M_{O\,x}^{ground} \\ M_{O\,y}^{ground} \\ M_{O\,z}^{ground} \end{pmatrix} + \begin{pmatrix} -Z_x \\ -Z_y \\ 0 \end{pmatrix} \times \begin{pmatrix} R_x^{ground} \\ R_y^{ground} \\ R_z^{ground} \end{pmatrix}\\ &= \begin{pmatrix} M_{O\,x}^{ground} - Z_y R_z^{ground} \\ M_{O\,y}^{ground} + Z_x R_z^{ground} \\ M_{O\,z}^{ground} + Z_y R_x^{ground} - Z_x R_y^{ground} \end{pmatrix}\end{aligned}$$

L'annulation des deux premières composantes donne lieu à deux équations dont les inconnues sont Z_x et Z_y. Ce système d'équations est régulier si et seulement si la composante R_z^{ground} est non-nulle, hypothèse raisonnable dans la phase de simple support du robot. Les coordonnées du ZMP sont dans ce cas :

$$ZMP = \begin{pmatrix} -\frac{M_{O\,y}^{ground}}{R_z^{ground}} \\ \frac{M_{O\,x}^{ground}}{R_z^{ground}} \\ 0 \end{pmatrix}$$

3.1.4 Le ZMP comme critère d'équilibre

Pour répondre à cette question, nous modélisons l'action du sol sur le pied par un champ de forces défini dans la partie S du plan correspondant à la zone de contact du pied. Nous notons

$$\mathbf{F}(x,y) = \begin{pmatrix} f_x(x,y) \\ f_y(x,y) \\ f_z(x,y) \end{pmatrix}$$

la force du sol sur le pied au point de coordonnées $(x, y, 0)$. Le moment des actions du sol sur le pied au ZMP s'exprime donc de la façon suivante :

$$\mathbf{M}_{ZMP}^{ground} = \int_S \vec{Z_{mp}M} \times \mathbf{f}(x,y) dS(M)$$

où $dS(M)$ est une mesure surfacique. Les composantes horizontales de ce moment s'écrivent :

$$\int_S (y - Z_y) f_z(x,y) dxdy = 0 \tag{3.13}$$

$$\int_S (Z_x - x) f_z(x,y) dxdy = 0 \tag{3.14}$$

$$\tag{3.15}$$

Supposons que le ZMP soit en dehors de l'enveloppe convexe de la surface d'appui (non nécessairement polygonale). Alors, il existe une droite qui sépare le ZMP de la surface d'appui. Au prix d'un changement de repère, on peut supposer que cette droite est l'axe $O\vec{y}$ du repère, que $Z_x < 0$ et que tous les points de la surface d'appui ont des abscisses positives. Le terme $(Z_x - x)$ de l'intégrale de l'équation (3.14) est donc uniformément négatif sur S et donc (3.14) ne peut être satisfaite que si f_z est uniformément nulle, condition exclue par nos hypothèses.

Ainsi, une condition nécessaire de stabilité du pied sur le sol est que le ZMP soit dans l'enveloppe convexe de la surface d'appui.

La position du ZMP peut également être calculée à partir de la dérivée du moment cinétique et de la résultante cinétique du robot. Exprimons les équations de la dynamique (3.11) et (3.12) au centre de masse du robot :

$$\begin{aligned} m\dot{\mathbf{v}}_G &= \mathbf{R}^{ground} + m\mathbf{g} \\ \dot{\sigma}_G &= \mathbf{M}_G^{ground} \end{aligned}$$

Exprimons maintenant le moment des forces du sol sur le robot en un point $Z = (x_Z, y_Z, 0)$ du sol :

$$\begin{aligned} \mathbf{M}_Z^{ground} &= \mathbf{M}_G^{ground} + \vec{ZG} \times (\mathbf{R}^{ground}) \\ &= \dot{\sigma}_G + m\vec{ZG} \times (\dot{\mathbf{v}}_G - m\mathbf{g}) \end{aligned}$$

En notant $\sigma_G = (\sigma_x, \sigma_y, \sigma_z)$ et (x_G, y_G, z_G) la position du centre de masse, l'équation précédente s'écrit :

$$\mathbf{M}_Z^{ground} = \begin{pmatrix} \dot{\sigma}_x \ + \ m\ (y_G - y_Z)\ (\ddot{z}_G + g) \ - \ m\ z_G\ \ddot{y}_G \\ \dot{\sigma}_y \ + \ m\ z_G\ \ddot{x}_G \ - \ m\ (x_G - x_Z)\ (\ddot{z}_G + g) \\ \dot{\sigma}_z \ + \ m\ (x_G - x_Z)\ \ddot{y}_G \ - \ m\ (y_G - y_Z)\ \ddot{x}_G \end{pmatrix}$$

Pour exprimer les coordonnées du ZMP, il suffit de résoudre le système d'équations obtenu en annulant les composantes x et y de $\mathbf{M}_Z^{ground}$. Nous obtenons alors :

$$\begin{aligned} x_{ZMP} &= x_G - \frac{\dot{\sigma}_y \ + \ m\ z_G\ \ddot{x}_G}{m\ (\ddot{z}_G + g)} \\ y_{ZMP} &= y_G + \frac{\dot{\sigma}_x \ - \ m\ z_G\ \ddot{y}_G}{m\ (\ddot{z}_G + g)} \end{aligned}$$

3.1.5 Génération d'une trajectoire de marche

La section précédente a construit incrémentalement, à partir des lois fondamentales de la dynamique, les relations nécessaires pour pouvoir générer une trajectoire de marche pour un robot humanoïde. Il s'agit désormais de choisir une stratégie de résolution qui ne soit non seulement correcte, mais qui puisse être mise en place sur le robot pour générer une trajectoire de marche stable.

Énoncé du problème La tâche à résoudre peut être énoncée ainsi : le robot doit aller du point A au point B de façon à préserver son équilibre. La stratégie la plus utilisée consiste en les étapes suivantes :

1. Planifier une série de points d'appui, c'est-à-dire d'empreintes de pas reliant les deux points. Les techniques classiques de planification de mouvement tel que la programmation dynamique – A* – ou bien encore les algorithmes probabilistes type RRT peuvent être utilisés dans ce cadre. En sortie, nous avons donc une suite d'empreintes de pas pour les deux pieds du robot annoté d'un moment de début et d'un moment de fin. Les moments où les deux pieds du robot sont au sol sont appelés phases de double support et les moments où un seul pied est au sol les phases de simple support.
2. Le ZMP est alors utilisé comme critère de stabilité, on définit alors une trajectoire de ZMP telle qu'à tout moment le ZMP est dans l'enveloppe convexe des points de contact. Une heuristique simple consiste à partir de la projection du centre de masse au sol au temps initial – le ZMP et le centre de masse sont confondus à l'arrêt –, puis de passer le ZMP sous la

cheville du pied gauche, réaliser un pas avec le pied droit, passer le ZMP sous la cheville du pied droit, réaliser un pas avec le pied gauche, etc. Il en ressort une trajectoire de référence pour le ZMP définie par une ligne brisée.

3. À partir de la trajectoire du ZMP, une trajectoire du centre de masse du robot doit être calculée. La section suivante détaillera ensuite comment suivre cette trajectoire du centre de masse pendant la marche.

Le problème reste alors la troisième étape : comment passer d'une trajectoire de ZMP à une trajectoire de centre de masse tout en préservant les contraintes temps réel qui sont inhérentes au contrôle d'un système robotique.

De la réalité physique à un modèle calculatoire tractable Le modèle décrit ici permet de passer des équations fondamentales de la physique, correctes, mais trop lourdes pour être directement prises en compte sur un robot humanoïde devant posséder un comportement réactif, c'est-à-dire ajuster son comportement au sein du contrôleur temps réel réalisant l'exécution sur une trajectoire de référence. Cette approche est décrite en détail dans Perrin et collab. (2010) mais nous rappellerons ici les grandes lignes de ces travaux. Tout d'abord, rappelons la formule du ZMP :

$$\begin{pmatrix} x_{\text{ZMP}} \\ y_{\text{ZMP}} \end{pmatrix} = \begin{pmatrix} x_G - \dfrac{\dot{\sigma}_y \ + \ m \ z_G \ \ddot{x}_G}{m \ (\ddot{z}_G + g)} \\ y_G + \dfrac{\dot{\sigma}_x \ - \ m \ z_G \ \ddot{y}_G}{m \ (\ddot{z}_G + g)} \end{pmatrix} \tag{3.16}$$

x_G, y_G, z_G étant la position du centre de masse dans l'espace, σ le moment d'inertie du robot, m la masse totale du robot et g la constante de gravité. L'objectif ici est de déduire x_G à partir du x_{ZMP} choisi. La résolution d'une telle équation est malheureusement difficile, car il n'existe pas de solution analytique et les solutions numériques sont lentes et nécessitent l'évaluation de plusieurs grandeurs, dont le calcul du moment d'inertie autour du centre de masse.

Cependant, cette équation peut devenir linéaire au prix des hypothèses suivantes :

– Moment d'inertie du centre de masse négligeable : $\dot{\sigma}_x = \dot{\sigma}_y = 0$
– Hauteur du centre de masse constante : z_G constant, $\ddot{z_G} = 0$

Ces hypothèses nous permettent d'obtenir les relations linéaires suivantes :

$$\begin{aligned} x_{\text{ZMP}} &= x_G - \frac{z_G}{g}\ddot{x}_G \\ y_{\text{ZMP}} &= y_G - \frac{z_G}{g}\ddot{y}_G \end{aligned}$$

Ce modèle dit du "Pendule Inverse Linéarisé" pose le problème sous la forme d'une équation différentielle non homogène du second ordre. La résolution explicite d'une telle équation est possible et peut s'exprimer sous la forme suivante :

$$\mathbf{x}(t) = \cosh(\sqrt{\frac{g}{z_G}}.t).\mathbf{V} + \sinh(\sqrt{\frac{g}{z_G}}.t).\mathbf{W} + \mathbf{r}(t) \tag{3.17}$$

$\mathbf{V}$ et $\mathbf{W}$ étant des constantes déterminées par les conditions aux limites du système, à savoir que le centre de masse atteigne la position fixée au début et à la fin avec une vitesse nulle. $\mathbf{r}(t)$ est un polynôme d'ordre 3, tout comme

la trajectoire du ZMP et dont les coefficients sont entièrement déterminés par elle. Les valeurs de $\mathbf{V}$ et $\mathbf{W}$ sont déterminées par les conditions aux limites du système :

$$\begin{pmatrix} \mathbf{x}_x(0) \\ \mathbf{x}_y(0) \end{pmatrix} = \begin{pmatrix} \mathbf{V}_x + \mathbf{r}_x(0) \\ \mathbf{V}_y + \mathbf{r}_y(0) \end{pmatrix} \tag{3.18}$$

$$\begin{pmatrix} \dot{x}(0) \\ \dot{y}(0) \end{pmatrix} = \begin{pmatrix} \frac{g}{z_G}.\mathbf{W}_x + \dot{\mathbf{r}}_x(0) \\ \frac{g}{z_G}.\mathbf{W}_y + \dot{\mathbf{r}}_y(0) \end{pmatrix} \tag{3.19}$$

Ces quatre variables déterminent la position et vitesse initiale du centre de masse. On a alors :

$$\begin{pmatrix} \mathbf{V}_x \\ \mathbf{V}_y \end{pmatrix} = \begin{pmatrix} \mathbf{x}_x(0) - \mathbf{r}_x(0) \\ \mathbf{x}_y(0) - \mathbf{r}_y(0) \end{pmatrix} \tag{3.20}$$

$$\begin{pmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{pmatrix} = \begin{pmatrix} \frac{z_G}{g}(\dot{\mathbf{x}}_x(0) - \dot{\mathbf{r}}_x(0)) \\ \frac{z_G}{g}(\dot{\mathbf{x}}_y(0) - \dot{\mathbf{r}}_y(0)) \end{pmatrix} \tag{3.21}$$

Cependant, nous n'avons pas de variable libre supplémentaire qui nous permettrait de contraindre la position et vitesse finale ($t = T$) de la trajectoire du centre de masse. Une stratégie pour contourner le problème peut être alors de remarquer qu'un changement de $\dot{\mathbf{x}}(0)$ fait varier $\mathbf{x}(T)$ de façon monotone. Il est alors possible de réaliser une recherche par dichotomie sur les valeurs de $\dot{\mathbf{x}}(0)$ afin de contraindre la position finale du centre de masse. Dans la mesure où l'on peut également adapter la longueur du pas T, il est possible de choisir T telle que les vitesses initiales et finales du centre de masse peuvent être négligées et la trajectoire du centre de masse considérée $\mathcal{C}^2$ sur la totalité de la trajectoire de marche.

Cette stratégie de calcul a pour avantage d'être simple à comprendre et à implémenter, mais également peu coûteuse en terme de temps de calcul. Elle a été publiée par Nicolas Perrin et al. dans Perrin N. (2011). Les limites de cette approche se situent dans les nombreuses hypothèses limitantes du modèle simplifié ainsi que par la recherche dichotomique des paramètres adaptés pour assurer la continuité de la trajectoire de centre de masse.

3.1.6 Exécution d'une trajectoire

La section précédente a montré comment générer, à partir d'une séquence d'empreintes de pas, une trajectoire de centre de masse qui assure la stabilité du robot. Une fois cette trajectoire associée aux trajectoires des pieds, on obtient une représentation d'une trajectoire de marche. Cette représentation est avantageuse, car elle ne fixe pas directement la position des corps du robot, il est alors encore possible de déplacer le haut du corps pour, par exemple, aller saisir un objet. Ce type de tâche étant typiquement définie comme un asservissement sur la donnée d'un capteur, il est nécessaire de pouvoir découpler ce qui relève de la planification et donc d'un processus lent et ce qui relève de l'asservissement capteur, c'est-à-dire d'un comportement réactif. On peut alors se demander, comment passer d'une série de trajectoires à une commande pouvant être envoyée aux actionneurs du système.

Ce problème peut se résoudre en tant que problème d'optimisation : l'objectif serait de minimiser la vitesse des actionneurs tout en ajoutant des contraintes pour suivre les trajectoires de référence. On pourrait imaginer avoir trois contraintes pour la trajectoire du centre de masse et des deux pieds. Il est alors possible d'ajouter encore une contrainte, pour atteindre un objet à empoigner, par exemple.

C'est à ce moment qu'une différenciation s'opère entre d'un côté les stratégies d'optimisation numérique classiques et les stratégies d'optimisation adaptées à la robotique de l'autre. Les problèmes que la robotique traite possèdent deux caractéristiques qui les rendent spécifiques :

1. Ils n'admettent pas toujours une solution : il est possible que l'objet soit trop loin pour pouvoir être atteint.
2. Les contraintes du problème ont une importance variable : suivre les trajectoires de référence sur les pieds et le centre de masse est critique, car ces contraintes permettent d'assurer la stabilité du robot. Échouer la saisie d'un objet est comparativement beaucoup moins important, car il sera toujours possible de retenter la saisie sans que cela mette le système dans une situation critique.

Enfin, le suivi de trajectoire et la génération de la commande doit être réalisée à une fréquence fixe. Le solveur doit donc pouvoir réaliser une itération durant un tour de la boucle temps réel ce qui impose des limitations très strictes sur la complexité des calculs pouvant être réalisés. Face à ces particularités, la stratégie de résolution adoptée ici est celle de la "pile de tâches".

Tâche et pile de tâches

Tâche L'objectif d'un contrôleur est de pouvoir calculer la commande à envoyer aux actionneurs qui va réaliser un certain nombre d'objectifs assignés au robot. Habituellement, il est mal aisé d'exprimer ses objectifs directement dans l'espace des commandes du système. À la place, nous pouvons exprimer ces objectifs sous forme de tâche. Chaque tâche vit dans un espace qui lui est propre et adapté à la contrainte que l'on souhaite exprimer. Par exemple, dans le cas d'un asservissement visuel, l'espace de la tâche sera l'espace image 2d. Une tâche d'asservissement visuel sera alors de réaliser un mouvement tel qu'un ensemble de points de l'image puisse atteindre une position prédéfinie.

Définition 31. *Une tâche est une fonction calculant l'erreur entre l'état courant d'un amer noté s et un état de référence prédéfini de cet amer noté $s*$:*

$$\mathbf{e} = \mathbf{s} - \mathbf{s}^* \tag{3.22}$$

L'objectif est donc de réaliser la tâche, c'est-à-dire de ramener l'erreur calculée par la tâche à zéro. L'erreur d'une tâche peut être de dimension supérieure à 1, dans ce cas l'objectif est de ramener toutes les composantes du vecteur d'erreur à 0. La définition de l'opérateur de comparaison entre les deux états est dépendant de l'espace de la tâche.

Jacobien La résolution d'une tâche passe par l'utilisation du jacobien de la tâche. Ce dernier est noté :

$$\mathbf{J}_i = \frac{\partial \mathbf{e}_i}{\partial \mathbf{q}} \tag{3.23}$$

Il lie l'espace des vitesses de la tâche à l'espace des vitesses articulaires du robot via la relation :

$$\dot{\mathbf{e}}_i = \mathbf{J}_i \dot{\mathbf{q}} \tag{3.24}$$

Pile de tâches On peut calculer la vitesse articulaire du robot en inversant la relation précédente :

$$\dot{\mathbf{q}} = \mathbf{J}_1^+ \dot{\mathbf{e}}_1^* + \mathbf{P}_1 \mathbf{z} \tag{3.25}$$

$\mathbf{J}_1^+$ est la pseudoinverse de $\mathbf{J}_1$ ou inverse des moindres carrés de $\mathbf{J}_1$. $\dot{\mathbf{e}}_1^*$, la consigne, est le mouvement souhaité dans l'espace de la tâche. $\mathbf{P}_1$ est le projecteur vectoriel sur le noyau du jacobien $\mathbf{J}_1$ et $\mathbf{z}$ un vecteur quelconque. Le projecteur est défini par :

$$\mathbf{P}_1 = \mathbf{I} - \mathbf{J}_1^+ \mathbf{J}_1 \tag{3.26}$$

Ce projecteur assure que, quel que soit le vecteur $\mathbf{z}$ choisi, il ne pourra pas interférer sur la réalisation de la consigne $\dot{\mathbf{e}}_1$. Par exemple, on peut prendre :

$$\mathbf{z} = \mathbf{J}_2^+ \dot{\mathbf{e}}_2^* \tag{3.27}$$

Dans ce cas, on aboutit à une commande telle que l'on essaie d'abord de réaliser la consigne $\dot{\mathbf{e}}_1^*$ puis $\dot{\mathbf{e}}_2^*$, sans perturber la réalisation de la première tâche. Cette pile de tâche possède deux niveaux de priorité, mais peut être généralisée à autant de niveaux de priorité que souhaité. Ce mécanisme de résolution est particulièrement adapté dans la mesure où l'on va d'abord résoudre la tâche de priorité la plus haute pour ensuite tenter de réaliser les secondes tâches avec les degrés de liberté restants. Si le problème n'est pas soluble dans sa totalité, les tâches possédant les priorités les plus élevées seront réalisées d'abord.

Différentes solutions sont possibles pour calculer la consigne, la plus simple consiste à utiliser une régulation proportionnelle :

$$\mathbf{s}_1^* = -\lambda(\mathbf{s}_1 - \mathbf{s}_1 *) \tag{3.28}$$

λ étant le gain du régulateur proportionnel. Les détails du solveur et de l'architecture utilisée dans le cadre de ces travaux sont détaillés dans Mansard et collab. (2009).

3.1.7 Limites de la boucle ouverte

Le schéma de contrôle consistant à simplement suivre une trajectoire planifiée n'est pas toujours satisfaisant. En effet, les nombreuses hypothèses réalisées dans le cadre de l'utilisation de modèles linéaires pour la marche entretiennent un flou entre les mouvements stables et instables sur une plate-forme donnée. En effet, les hypothèses telles que négliger les moments autour du centre de masse, supposer un sol plan, ne sont jamais respectées en réalité. Il faut donc pouvoir apprécier ce qui est une entorse "acceptable" aux hypothèses sur lesquelles le raisonnement a été effectué. Il en découle une grande part de "savoir-faire" dans

l'implémentation d'un algorithme de marche fiable. Et même avec ce savoir-faire, l'écart entre le modèle et la réalité est tel qu'il met souvent en péril la réalisation de tâches complexes. Lors de l'utilisation d'un modèle linéaire, l'erreur de positionnement du robot peut augmenter d'un ou deux centimètres par pas. Dès lors, prendre un objet ou interagir avec un humain se révèle impossible après un déplacement de quelques mètres sans un mécanisme de compensation adapté. Inversement, l'utilisation d'un modèle physique réaliste aboutit à des temps de calcul importants, trop longs pour pouvoir être utilisés pour s'asservir sur un capteur notamment. La recherche d'un compromis entre réactivité, fiabilité et précision est un réel enjeu pour la robotique humanoïde.

3.1.8 Conclusion

Cette section propose une stratégie de génération de mouvement pour les robots humanoïdes de l'État de l'Art. Nous sommes partis des fondements de la robotique avec la définition d'un arbre cinmatique, de la cinématique directe et inverse pour ensuite détailler comment les relations fondamentales de la physique ont permis de définir des modèles de génération de mouvement tractables pour le calcul de trajectoire de référence pour la marche. On a enfin montré que ces trajectoires pouvaient être suivies via un contrôleur temps réel reposant sur le paradigme de la pile des tâches. L'utilisation de tâches permet la conception de contrôleurs versatiles, pouvant être étendus facilement par la définition de nouvelles tâches. Cette stratégie globale relève de nombreux travaux du domaine réalisés qui seront détaillés dans la section suivante.

3.2 État de l'Art

Cette section réalise un état de l'Art sommaire en robotique humanoïde. Il comprend à la fois une analyse des travaux de référence en robotique, de différents travaux sur lesquels l'algorithme de contrôle proposé dans ce chapitre s'appuie ainsi que les différents efforts qui ont été réalisés jusqu'alors pour parvenir à exécuter des tâches complexes sur les robots bipèdes malgré la dérive induite par ce mode de locomotion.

3.2.1 Travaux de référence en robotique

La littérature robotique contient plusieurs livres de référence détaillant les fondements du domaine. Un exemple remarquable est Siciliano et Khatib (2008) détaillant la plupart des algorithmes fondamentaux du domaine et possédant un chapitre consacré à la locomotion bipède. Dans le domaine de la localisation, on peut également citer Thrun et collab. (2005), pour la vision robotique Hartley et Zisserman (2003) et pour la planification de mouvements Latombe (1991); Choset et collab. (2005); LaValle (2006). Les fondements mathématiques de la robotique sont clairement décrits par Murray et collab. (1994).

3.2.2 Modèles pour la marche dynamique

Le développement de la locomotion bipède est étroitement lié avec la recherche de modèles simplifiés permettant de générer des mouvements stables

en un temps raisonnable. En particulier, la génération de mouvements utilisant le ZMP comme critère de stabilité a d'abord été étudié par le chercheur serbe Miomir Vukobratović – 1er octobre 1931, 11 mars 2012 –. Il a publié en 2004 Un bilan des recherches sur l'utilisation du ZMP pour la robotique Vukobratovic et Borovac (2004). Un second chercheur particulièrement reconnu pour le développement de stratégies de génération de mouvement réactif fondé sur le ZMP est Shuuji Kajita – 梶田　秀司 –. Une sélection des travaux des équipes japonaises du domaine est Harada et collab. (2004); Kajita et collab. (2001); Morisawa et collab. (2007).

Des recherches ont également été menées pour formuler le problème de génération de mouvement à l'aide de solveurs supportant les contraintes inégalités Dimitrov et collab. (2011); Herdt et collab. (2010) afin de rechercher la trajectoire du centre de masse du robot parmi toutes les solutions correctes au lieu de spécifier une trajectoire de ZMP pour en déduire une trajectoire de centre de masse solution. La formulation générique du problème a également permis d'inclure le positionnement des points de contact pendant la marche comme partie intégrante du problème. L'utilisation du démarrage à chaud Dimitrov et collab. (2009) a également permis des gains en terme de temps de calcul non négligeables.

Alors que ces équipes ont focalisé leurs efforts sur la rapidité des algorithmes, d'autres ont tenté de concevoir des modèles plus proches des modèles physiques classiques afin de générer des mouvements dynamiques complexes. Un exemple est Mombaur (2009) qui traite de la génération de trajectoires de course. D'autres travaux incluent Suleiman et collab. (2007); Kanehiro et collab. (2008).

3.2.3 Contrôleurs pour robots humanoïdes

Le formalisme du contrôle par tâche a notamment été étudié par Bernard Espiau Rives et collab. (1989), ainsi que d'autres groupes tel que celui de Luis Sentis Sentis et Khatib (2005). Les publications sur l'approche de contrôle par tâche dévéloppé au sein du de notre groupe de recherche sont Mansard et collab. (2009); Yoshida et collab. (2005); Kanoun et Laumond (2010).

3.2.4 Exécution de trajectoires asservies

Si la robotique mobile classique a dédié un grand nombre de publications au problème de la navigation Samson et Ait-Abderrahim (1991); Lefebvre et collab. (2005), la littérature est beaucoup moins abondante dans le domaine de la robotique bipède. Des exemples de scénarii complexes sur des systèmes humanoïdes sont Michel et collab. (2005); Chestnutt (2010); Chestnutt et collab. (2009). Aucun traitement particulier pour la compensation de la dérive n'est pas décrite dans ces travaux bien qu'elle n'a pu être ignorée au vu des possibilités des systèmes décrits. D'autres papiers s'intéressent plus spécifiquement au maintien de l'équilibre dans le cas où une perturbation extérieure apparaît lors de la marche Morisawa et collab. (2007).

3.3 Correction de mouvement temps réel

3.3.1 Suivi de trajectoires : des robots mobiles aux humanoïdes

Le problème du suivi de trajectoire a été largement étudié et l'utilisation d'une technique classique de la littérature robotique telle que celle illustrée par la figure 3.3 semble naturelle. Cependant, nous allons montrer qu'appliquer les stratégies de la robotique mobile n'est pas suffisant pour réaliser un suivi de trajectoire référencé capteur sur un robot humanoïde.

La figure 3.3 montre comment peut fonctionner un contrôleur réalisant un asservissement capteur pour le suivi de trajectoire sur un robot mobile. Un système de localisation externe fournit une appréciation de la position du robot qui permet de déformer continûment la trajectoire de référence γ. L'estimation de la position du robot par le système de localisation est dénotée $\hat{\mathbf{x}}$. De cette estimation, une erreur sur le positionnement peut être calculée et ajoutée composante par composante à la trajectoire de référence en utilisant, par exemple, un proportionnel, c'est-à-dire k_x, k_y, k_{r_z} sur la figure.

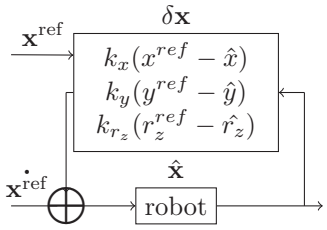

FIGURE 3.3 – Correction de trajectoire pour un robot type base mobile. $\mathbf{x}^{\text{ref}} = (x^{\text{ref}}, y^{\text{ref}}, r_z^{\text{ref}})$, $\dot{\mathbf{x}}^{\text{ref}} = (\dot{x}^{\text{ref}}, \dot{y}^{\text{ref}}, \dot{r_z}^{\text{ref}})$ et $\hat{\mathbf{x}} = (\hat{x}, \hat{y}, \hat{r_z})$ sont respectivement la position et vitesse du robot dans le plan et dans la réalité. La commande planifiée $\dot{\mathbf{x}}$ est corrigée en sommant $\delta\mathbf{x}$ une correction déterminée par l'erreur de positionnement d'un point de référence du robot entre le plan et la position donnée par le système de localisation.

Si l'on appliquait ce système à un robot humanoïde, il mettrait à jour une trajectoire de référence, celle du centre de masse, par exemple. Tant que le robot a au moins un pied au sol, le bassin est contrôlable localement ce qui permet d'appliquer la correction en modifiant la valeur de degrés de liberté de la jambe.

Stabilité de la trajectoire corrigée Contrairement aux robots mobiles, les trajectoires stables pour un robot humanoïde sont peu nombreuses en proportion. Tout l'enjeu de la génération d'une trajectoire pour un robot humanoïde est d'assurer sa stabilité. Il faut donc vérifier que la correction de la trajectoire initiale ne viendra pas compromettre cette stabilité. Comme nous l'avons vu dans la section précédente, il faut corriger la trajectoire sans faire sortir le ZMP du polygone support :

$$\mathbf{z} = \mathbf{x} - \frac{z_c}{g}\ddot{\mathbf{x}} \tag{3.29}$$

La linéarité de l'équation signifie que perturber le centre de masse par une fonction $\delta\mathbf{x}$ dépendant du temps perturbe le ZMP de manière linéaire :

$$\begin{aligned}\mathbf{z}' &= (\mathbf{x}+\delta\mathbf{x}) - \frac{z_c}{g}.\frac{d^2(\mathbf{x}+\delta\mathbf{x})}{dt^2} \\ &= \mathbf{x} - \frac{z_c}{g}\ddot{\mathbf{x}} + \underbrace{\delta\mathbf{x} - \frac{z_c}{g}\ddot{\delta\mathbf{x}}}_{\text{perturbation induite}}\end{aligned} \tag{3.30}$$

En utilisant le modèle linéaire, il est possible d'étudier comment modifier la trajectoire du centre de masse pour assurer l'équilibre de la trajectoire.

De plus, il est possible d'effectuer le raisonnement en utilisant le modèle linéaire sur la perturbation uniquement. On peut donc utiliser un algorithme de génération de mouvement hors-ligne coûteux et raffiner le mouvement en temps réel via la méthode proposée dans la section suivante. L'intérêt est de pouvoir combiner à la fois des trajectoires qui sont trop dynamiques pour pouvoir utiliser le modèle linéaire tout en considérant localement que les équations sont linéaires pour ajuster localement la trajectoire.

Singularités des trajectoires corrigées Dans la mesure où le bassin d'un robot humanoïde n'est que localement actionnable, il est primordial de générer des corrections qui n'introduisent pas de singularités durant le mouvement. Typiquement, un écart trop important entre les points de contact et la trajectoire du centre de masse – souvent proche du bassin – peut provoquer des singularités. Il est donc nécessaire non seulement de corriger le positionnement du bassin, mais également de déformer les points de contact afin d'éviter les singularités. Dans le cas contraire, on observerait une "dérive" du bassin qui s'éloignerait progressivement des empreintes de pas planifiées.

Il apparaît donc clairement qu'une solution naïve inspirée de la robotique mobile n'est pas une solution viable. Dès lors, corriger la trajectoire d'un robot humanoïde pour pouvoir s'asservir sur un capteur demande une approche originale qui sera détaillée dans la section suivante. Le schéma de contrôle proposé tient compte de ces deux problèmes afin de fournir une approche adaptée à la locomotion bipède.

3.4 Suivi de trajectoire

Cette section propose un contrôleur pour le suivi de trajectoire boucle fermée pour les robots humanoïdes qui prend en compte leurs particularités.

Suivre une trajectoire asservie consiste à exécuter une trajectoire précalculée tout en s'assurant que les erreurs d'exécution sont bien compensées. Un tel système se divise en quatre composants :

1. un générateur de trajectoire,
2. un système de localisation fournissant une estimation de la position du robot,
3. un système d'estimation de l'erreur calculant l'écart entre la position actuelle du robot et celle à laquelle il est censé se trouver,

4. et un composant générant une trajectoire mise à jour permettant de réduire l'écart constaté.

Le générateur de trajectoires fournit deux données de référence : la séquence d'empreintes de pas, c'est-à-dire un ensemble de S_i tel que $0 \leq i \leq n^{\text{pas}}$ et une trajectoire du haut du corps $\gamma(t \in [t_{\min}, t_{\max}]) \in \mathcal{C}$. Un avantage du schéma de contrôle proposé est que la position des empreintes de pas peut être altérée pour s'assurer que la trajectoire reste faisable. En partant d'une modification de la pile de pas, on peut déformer la trajectoire du centre de masse pour s'assurer que la trajectoire reste stable. Les trajectoires de référence portent sur des points du corps du robot et ne contiennent pas directement les valeurs articulaires. La pile de tâches a donc pour tâche de résoudre la cinématique inverse en temps réel lors de la résolution du problème d'optimisation.

Une itération de la boucle de contrôle est décrite par l'algorithme 1 et peut être résumée aux étapes suivantes :

1. estimation de la position du robot,
2. calcule de l'erreur de positionnement $\delta\mathbf{x}$,
3. filtrage de l'erreur calculée par un filtre passe-bas afin de borner la déformation maximum et absorber d'éventuelles erreurs de localisation,
4. recalcule les prochains pas de façon à absorber l'erreur de positionnement du robot,
5. vérifie si le prochain pas est réalisable,
6. regénère des trajectoires continues pour les pieds, le ZMP et le centre de masse,
7. recalcule les valeurs articulaires via la pile de tâches

Algorithme 1 Boucle de contrôle au temps $t_{\text{maintenant}}$ réalisant un suivi de trajectoire boucle-fermée de la trajectoire γ (la prochaine correction sera appliquée au temps t_{prochain}). $\bigoplus$ étant l'opération usuelle du groupe considéré – SE(3) pour les trajectoires de pied et $\mathbb{R}^2$ pour la trajectoire du centre de masse.

Input: γ, $t_{\text{maintenant}}, t_{\text{prochain}}$
Output: γ, $t_{\text{maintenant}}, t_{\text{prochain}}$
  if $\gamma(t_{\text{maintenant}})$ est en double support **and** $t_{\text{maintenant}} \geq t_{\text{prochain}}$ **then**
    estimer la position du robot $\hat{\mathbf{x}}$
    calculer l'erreur de positionnement $\delta\mathbf{x}$
    calculer la déformation $\delta\gamma$ permettant de corriger l'erreur de positionnement $\delta\mathbf{x}$
    if la correction $\delta\gamma$ peut être appliquée **then**
      $\forall t \in [t_{\text{maintenant}}, t_{\max}], \gamma(t) \leftarrow \gamma(t) \bigoplus \delta\gamma(t)$
      $t_{\text{prochain}} \leftarrow t_{\text{maintenant}} + 2T_{\text{step}}$
    end if
  end if
  $\mathbf{q} \leftarrow \gamma(t_{\text{maintenant}})$
  $t_{\text{maintenant}} \leftarrow t_{\text{maintenant}} + 2T_{\text{step}}$

Tout d'abord, la position du robot $\hat{\mathbf{x}}$ est déterminée. Une large littérature concernant la localisation robotique existe Stasse et collab. (2008b); Thompson

et collab. (2006) et les spécificités de la localisation d'un robot humanoïde seront abordées dans le prochain chapitre. Les limites habituelles de la localisation sont la perte du suivi résultant dans l'absence de localisation pendant un moment plus ou moins long ou bien la présence de bruit, voire de valeurs aberrantes dans l'estimation fournie.

Deuxièmement, une erreur $\delta\mathbf{x}$ est calculée en comparant la position planifiée et la position estimée du corps de référence. Un seuil est appliqué pour borner la correction pouvant être appliquée. En pratique, ce filtrage permet de limiter l'influence du bruit de localisation ainsi que les valeurs aberrantes lorsque la localisation échoue.

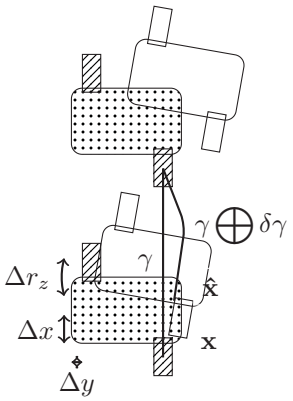

FIGURE 3.4 – Correction du prochain pas suite à une erreur d'exécution. La position planifiée du bassin x est en pointillé – avant et après le prochain pas –. La position estimée du robot $\hat{\mathbf{x}}$ est représentée en trait plein. L'erreur selon les axes X, Y et l'erreur angulaire sont $(\Delta x, \Delta y, \Delta r_z) = \delta\mathbf{x}$ et $\delta\gamma$ est la trajectoire corrigée atteignant l'empreinte de pas planifiée.

Ensuite, la position du prochain pas est modifiée afin de compenser l'erreur calculée. La figure 3.4 illustre cette étape.

À partir de ce moment, il est possible de regénérer des trajectoires pour les pieds et le centre de masse. Afin d'assurer la continuité de la solution, la correction de la trajectoire est appliquée progressivement durant les deux prochains pas. Pour finir, la résolution des tâches calcule les nouvelles valeurs articulaires.

Avant de commencer une correction, un test est effectué pour déterminer si une correction est possible au moment courant $t_{\text{maintenant}}$. Une correction ne peut commencer que si le robot est en double support et qu'aucune autre correction n'est en cours de réalisation, c'est-à-dire $t_{\text{maintenant}} \geq t_{\text{prochain}}$. En effet, corriger de nouveau alors que la correction précédente n'est pas totalement appliquée conduit à corriger une partie de l'erreur de localisation plusieurs fois.

Les travaux précédents tel que Harada et collab. (2004); Morisawa et collab. (2007) ont pour but de permettre des changements soudains dans la trajectoire des pieds – et donc du centre de masse – tout en préservant l'équilibre du robot. Le schéma de contrôle proposé a pour but, au contraire, de rester aussi proche que possible du plan initial et ne sert donc pas le même objectif.

Estimation de l'erreur de positionnement Nous faisons l'hypothèse ici que le système fournit $\hat{\mathbf{x}} \in \mathrm{SE}(2)$, une estimation de la position actuelle du robot.

Si $\mathbf{x} \in \mathrm{SE}(2)$ est la position planifiée du robot et $\hat{\mathbf{x}} \in \mathrm{SE}(2)$ sa localisation estimée, l'erreur est définie par :

$$\delta\mathbf{x} = \mathbf{x}.\hat{\mathbf{x}}^{-1} \tag{3.31}$$

Dans l'équation précédente, $\delta\mathbf{x}(t)$ peut être interprétée comme la position du robot planifiée par rapport à la position du robot estimée. Le délai induit par le système de localisation fait que l'estimation du robot doit être comparée à la position dans le plan au même moment. On obtient alors une erreur sur le positionnement à un moment du passé, mais qui peut être utilisée sans problème. On peut raisonnablement estimer que l'erreur de positionnement croît avec le temps et qu'on ne compensera donc l'erreur qu'en partie, mais on évitera une trop grande divergence entre le plan et la réalité. On estime également que l'erreur de positionnement initial est toujours nulle :

$$\delta\mathbf{x}(t_{\min}) = 0 \tag{3.32}$$

Modification de la séquence de pas Étant donné une erreur de localisation d'un corps de référence, le bassin en général, il est possible d'altérer la pile de pas restants à exécuter pour absorber cette erreur.

L'objectif de cette étape est donc de faire en sorte de déformer la suite du mouvement pour que le robot pose ses pieds aux endroits planifiés.

Les empreintes de pas sont $S \in \mathrm{SE}(2)^{n^{\text{pas}}}$. Considérons $\delta\mathbf{x}$, l'erreur de positionnement courant, et $S^{\text{futur}} \subset S$ les pas restant encore à réaliser. Les empreintes de pas futures seront déformées en utilisant la relation suivante :

$$\forall s \in S^{\text{future}}, s \leftarrow \delta\mathbf{x}.s \tag{3.33}$$

Modification des trajectoires de référence De nouvelles positions pour les points d'appui ont été calculées. Il est donc désormais nécessaire de modifier la trajectoire des deux pieds ainsi que du centre de masse pour pouvoir corriger le positionnement des deux pieds dans le futur.

La correction de la trajectoire du centre de masse est calculée en utilisant le modèle simplifié décrit par 3.17. De fait, aucune hypothèse n'est faite pour réaliser la correction et en particulier le générateur de trajectoire n'est jamais mis à contribution pour regénérer les trajectoires de référence. C'est là la différence majeure qui différencie cette approche de la replanification. Ces approches se fondent sur un recalcul complet des trajectoires de marche régulière en changeant la position du robot à partir de l'estimation de sa position. Si cette approche est adaptée lorsque la planification a un coût peu important, comme sur les bases mobiles, elle ne l'est pas dans le cadre de la robotique humanoïde où les algorithmes de génération de mouvement ont un coût important. L'hypothèse proposée ici est qu'utiliser un modèle linéarisé pour réaliser de petites corrections perturbe peu la trajectoire et ne compromet pas la qualité du résultat de manière significative.

Supposons que le modèle linéaire ait été utilisé pour générer le mouvement, la résolution analytique nous donne l'équation suivante, déjà détaillée dans la section précédente :

$$\mathbf{x}(t) = \cosh(\sqrt{\frac{g}{z_G}}.t).\mathbf{V} + \sinh(\sqrt{\frac{g}{z_G}}.t).\mathbf{W} + \mathbf{r}(t) \tag{3.34}$$

Dans la section précédente, nous avons déplacé les pas futurs. L'objectif est donc de raccrocher la trajectoire de ZMP à la nouvelle pile de pas. Cependant, contrairement aux pas qui sont des données discrètes, les trajectoires doivent être raccordées continûment. Pour ce faire, nous avons choisi un polynôme de degré trois à valeurs réelles auquel quatre contraintes ont été imposées. Ce polynôme est noté $\Delta_\alpha(t)$ où α est la valeur du polynôme à $t = T$. Les contraintes sont donc :

1. $\forall t \leq 0, \Delta_\alpha(t) = 0$
2. $\forall t \leq 0, \dot{\Delta_\alpha}(t) = 0$
3. $\forall t \geq T, \Delta_\alpha(t) = \alpha$
4. $\forall t \geq T, \dot{\Delta_\alpha}(t) = 0$

En pratique, T représente la durée d'un pas. Ce polynôme sera utilisé pour fournir des transitions continues, composante par composante, aux différentes trajectoires de référence. Qui plus est, les quatre contraintes contraignent totalement ce polynôme de degré trois pour lequel il existe donc une unique solution sur l'intervalle $[0, T]$. Les valeurs extérieures étant définies par les contraintes directement. On notera enfin que même si ce polynôme est à valeur dans $\mathbb{R}$ quand $\alpha \in \mathbb{R}$, on peut aussi bien le définir sur $\mathbb{R}^n$, $n \in \mathbb{N}^+$ à condition que $\alpha \in \mathbb{R}^n$.

Une fois ce polynôme défini, il est possible de rattacher la trajectoire de ZMP en sommant à la trajectoire de référence du ZMP, le polynôme $\mathbf{\Delta}_{\delta\mathbf{x}}(t)$. La résolution du système est alors le suivant :

$$\bar{\mathbf{c}}(t) = \cosh(\sqrt{\frac{g}{z_c}}.t).\mathbf{V} + \sinh(\sqrt{\frac{g}{z_c}}.t).\mathbf{W} + \mathbf{r}(t) + \mathbf{\Delta}_{\delta\mathbf{x}}(t) \tag{3.35}$$

En effet, la substitution de variable $\mathbf{r}(t)$ par $\mathbf{r}(t) + \mathbf{\Delta}_{\delta\mathbf{x}}(t)$ ne change que le second membre de l'équation homogène et ne perturbe pas la solution générale.

L'équation précédente réalise une correction à $t = 0$. Évidemment, ce n'est pas toujours le cas. En pratique, une correction est commencée tous les deux pas, lorsque le robot est en double support. Si l'on a corrigé trois fois la trajectoire à t_1, t_2 et t_3, la trajectoire est alors :

$$\bar{\mathbf{c}}(t) = \cosh(\sqrt{\frac{g}{z_c}}.t).\mathbf{V}+\sinh(\sqrt{\frac{g}{z_c}}.t).\mathbf{W}+\mathbf{r}(t)+\mathbf{\Delta}_{\delta\mathbf{x}(t_1)}(t-t_1)+\mathbf{\Delta}_{\delta\mathbf{x}(t_2)}(t-t_2)+\mathbf{\Delta}_{\delta\mathbf{x}(t_3)}(t-t_3) \tag{3.36}$$

$\delta\mathbf{x}(t)$ étant l'erreur de localisation au temps t. Plus généralement, on obtient donc la relation suivante où $t_{\text{correction}}$ est l'ensemble des temps où une correction a été appliquée :

$$\bar{\mathbf{c}}(t) = \cosh(\sqrt{\frac{g}{z_c}}.t).\mathbf{V}+\sinh(\sqrt{\frac{g}{z_c}}.t).\mathbf{W}+\mathbf{r}(t)+\sum_{t_i \in t_{\text{correction}}} \mathbf{\Delta}_{\delta\mathbf{x}(t_i)}(t-t_i) \tag{3.37}$$

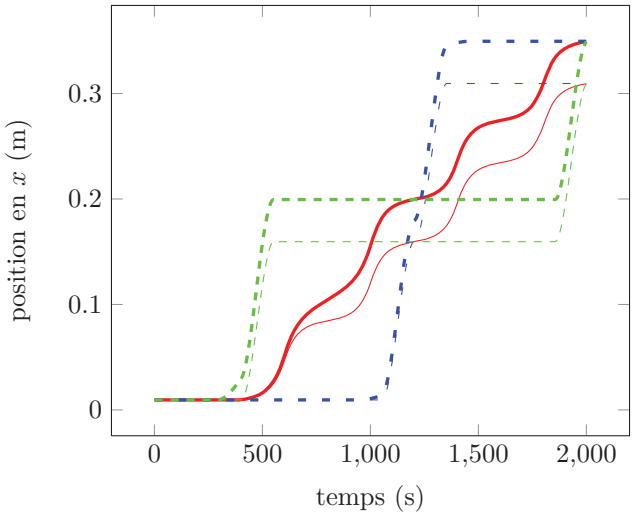

FIGURE 3.5 – Évolution, pendante deux pas, de la position en x du centre de masse (——), du pied gauche (- -) et du pied droit (- - -). La courbe en gras illustre un pas de 0.3m en avant. Les courbes en pointillé illustre la position du centre de masse (——), pied gauche (- -) et pied droit (- - -) après un pas de 0.05m en avant, soit une correction de 0.02m.

L'équation précédente définit comment déformer la trajectoire du centre de masse. La déformation de la trajectoire des pieds est directe, en utilisant le polynôme $\mathbf{\Delta}$ pour assurer que les trajectoires de chaque pas terminent bien à la position prévue dans la pile de pas modifiée. Cependant, un dernier point reste à traiter : la synchronisation temporelle de la correction du pied gauche, droit et du centre de masse. Elle est réalisée en deux temps : le premier pied qui bouge et le centre de masse sont d'abord corrigés puis le deuxième pied est ensuite corrigé pendant sa phase de vol. Évidemment, il n'est pas possible de déplacer un pied au sol d'où la nécessité d'utiliser deux pas pour corriger complètement la trajectoire. On remarquera que la modification synchronisée de la trajectoire du centre de masse et de la trajectoire du premier pas permet d'assurer que le ZMP reste à tout moment dans le polygone support. En effet, la trajectoire du ZMP étant déformée d'autant que la trajectoire du centre de masse, ce dernier termine à une position équivalente à la fin du pas, relativement au pas ayant effectué le mouvement. La Figure 3.5 illustre ces trois corrections synchronisées.

Filtrage de l'erreur et faisabilité du prochain pas La correction ne compromet pas la stabilité du robot. Cependant, la correction n'est pas toujours réalisable, car l'espace d'accessibilité du pied est fini à la fois à cause de sa géométrie et d'éventuelles autocollisions pouvant se produire – une jambe pouvant entrer en collision avec la seconde –. De ce fait, il est important de borner les corrections pour éviter les autocollisions.

Les hypothèses de travail sont que la trajectoire de référence est elle-même stable et sans collision. Par conséquent, l'objectif est alors de savoir quelles sont les bornes à partir desquelles il n'est plus possible de corriger le mouvement.

La première étape a été de déterminer une enveloppe dans laquelle les corrections sont réalisables géométriquement parlant – la cinématique inverse des jambes est soluble –. La perturbation latérale maximum est de ±0.04m, la perturbation frontale, quant à elle, est de ±0.05m et la perturbation angulaire maximum est de ±0.1rad tous les deux pas. Ces valeurs correspondent à une borne sur la valeur $\delta\mathbf{x}$, l'erreur de positionnement du robot.

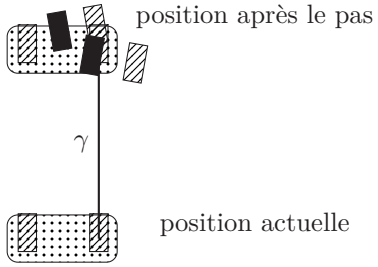

FIGURE 3.6 – Validation du pas recalculé. La position du bassin est symbolisée par le rectangle en pointillé. Les pas valides sont hachurés tandis que les pas invalides sont de couleur noire.

Ensuite, le nouveau pas est validé. La décision se fonde sur le raisonnement géométrique suivant : le pied d'HRP-2 ne peut pas entrer en collision avec lui-même. Par conséquent, les seules collisions possibles sont qu'un pied entre en collision avec le second. De ce fait, si l'on ignore les configurations rares du type, "le robot croise les jambes en marchant", possibles, mais aux limites des capacités articulaires des jambes, on peut tenir le raisonnement suivant : on peut toujours déplacer un pas du pied gauche vers la gauche sans entrer en collision et un pas du pied droit vers la droite sans entraîner de collision. De même on peut toujours faire tourner la structure vers l'extérieur sans provoquer de collisions. L'heuristique suivante est donc utilisée pour valider les pas. Elle est, évidemment, sous-optimale, mais est sûre et a un coût en temps de calcul quasi nul.

Si la correction est invalidée pour une des deux raisons précédemment décrites, elle est alors abandonnée. À la prochaine phase de double support, une nouvelle correction sera calculée et une nouvelle phase de validation sera tentée. Dans la mesure où l'erreur de positionnement croît lentement, il y a alors deux cas :

- Dans le premier, le signe de l'erreur de positionnement $\delta\mathbf{x}$ a changé, ce qui signifie que l'erreur de localisation était faible et qu'il n'était pas critique de la corriger maintenant,
- Dans le second, le signe de l'erreur reste constant et il sera possible de corriger l'erreur, car la direction dans laquelle la correction est possible s'inverse en passant d'un pas avec le pied gauche à un pas avec le pied droit.

Ce système empirique a été validé avec succès sur le robot HRP-2. Un objectif pour le futur est de remplacer cette heuristique par une validation complète de la trajectoire de pas modifiée. Une stratégie telle que celle décrite dans Perrin et collab. (2010) pourrait alors être envisagée.

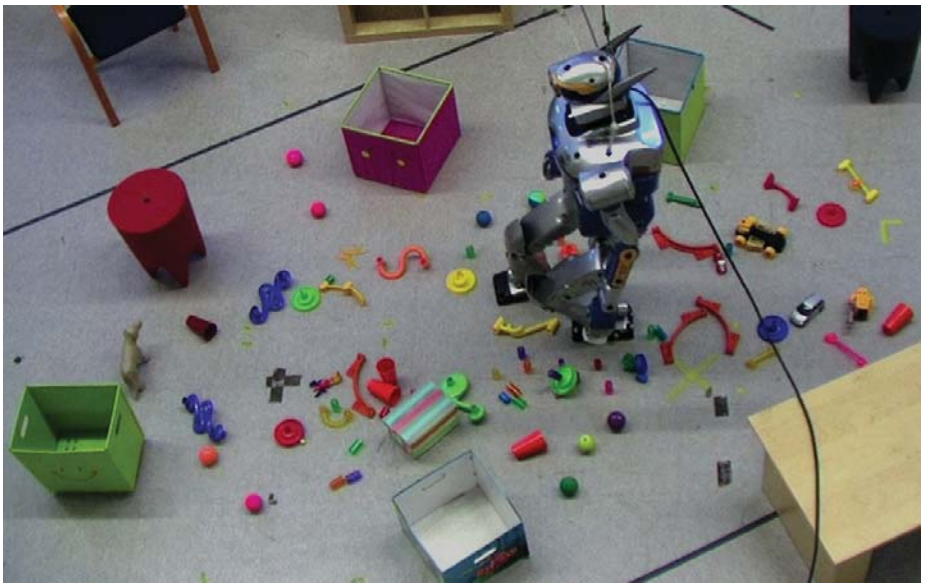

FIGURE 3.7 – HRP-2 marchant dans un environnement extrêmement encombré tout en évitant des obstacles. La position finale est atteinte avec une précision de ±3cm. L'erreur finale est la conséquence du bruit impactant l'estimation de la position du robot ainsi que de la dérive accumulée sur la fin de la trajectoire qu'il n'est pas possible de corriger, faute de temps.

3.5 Résultats expérimentaux

La méthode proposée a été validée sur la plate-forme HRP-2 grâce au scénario suivant : le robot doit naviguer dans un environnement particulièrement contraint tout en enjambant des obstacles. La longueur de la trajectoire est approximativement de 2.5m et est exécutée en environ 40s.

Ce scénario démontre que l'utilisation de ce schéma de contrôle permet une navigation fiable. Le robot atteint l'objectif fixé avec une précision de quelques centimètres tandis que le positionnement final en utilisant un schéma de contrôle en boucle ouverte a une erreur de plus ou moins 50 centimètres. De plus, ce schéma de contrôle est robuste aux pertes de suivi et à des estimations peu précises par le système de localisation. En effet, l'expérience s'appuie sur un système de capture de mouvement pour localiser le robot. Ces derniers sont extrêmement précis – précision au millimètre dans les cas favorables – mais nécessitent un environnement dégagé ce qui n'est pas le cas ici. Les marqueurs posés sur le pied sont parfois occlus par les obstacles ce qui fait varier la précision durant le mouvement. Ce comportement ne perturbe pas le schéma de correction qui peut atteindre la position finale sans problème.

La vidéo de l'expérience est disponible sur internet :

`http://youtu.be/cUZOnNiPs70`

La figure 3.7 fournit un aperçu de l'expérience réalisée. Le robot part de la droite et marche au travers des obstacles jusqu'à arriver à son but, sur la partie gauche de l'image.

Dans ce contexte, la précision atteinte est de ±2cm. Cette trajectoire a été jouée cinq fois consécutivement sans donner lieu à une collision avec un obstacle ou à une autocollision. En utilisant un mouvement réalisant de plus petits pas, la précision peut atteindre 1cm. Les petits pas nécessitent une accélération moindre du haut du corps et les effets dynamiques mal modélisés par le modèle linéaire se font moins ressentir, il en découle une erreur de positionnement des empreintes de pas plus faible. L'algorithme de correction pour sa part dispose de davantage de pas pour pouvoir réaliser des corrections. Ces deux facteurs jouant pour améliorer la précision du suivi de trajectoire.

3.6 Conclusion

Cette section est divisée en deux et sert deux objectifs distincts. La première partie est une introduction à la robotique humanoïde et explique, en partant des relations fondamentales de la physique, certains résultats récents de la littérature afin de fournir tous les outils permettant de réaliser une tâche de locomotion pour un robot humanoïde. Les robots humanoïdes alliant à la fois sous-actionnement et redondance nécessitent des stratégies adaptées afin de pouvoir générer des mouvements dans un temps raisonnable. En effet, le problème en robotique humanoïde n'a jamais été de prévoir le mouvement et la dynamique de corps dont les mouvements relatifs sont contraints. Ces équations fondamentales de la physique sont connues depuis longtemps. Le véritable enjeu est de trouver des modèles calculatoires adaptés permettant un compromis idéal entre réalisme, expressivité et réactivité. Différents outils de l'État de l'Art ont été détaillés et sont assemblés pour former une plate-forme robotique cohérente.

La seconde partie propose un contrôleur pour le suivi de trajectoire boucle fermée afin d'autoriser la navigation de précision pour le robot humanoïde HRP-2. Cette section montre la barrière à franchir pour passer d'une trajectoire simulée qui semble réalisable à une application robotique fonctionnelle où la réalité physique et l'incertitude d'exécution inhérente à la robotique réelle jouent un rôle prépondérant. Nous avons essayé de montrer que des mouvements complexes nécessitent des algorithmes complexes qui seront difficiles, même à terme, à passer dans les contrôleurs temps réels des robots. De ce fait, si l'on veut pouvoir dépasser la simple marche dans un sol plan, il faudra sans doute passer par des stratégies d'adaptation locales et réactives du plan plutôt que par une replanification globale, ce que les approches récentes tendent à réaliser.

4 Primitives de mouvement

> Vous n'avez qu'à marcher de vertus en vertus.
>
> Jean Racine
> *Britannicus*

4.1 Problématique

Le chapitre précédent a introduit la possibilité d'asservir une trajectoire de marche sur un robot humanoïde. Cependant, les tâches accomplies par les robots humanoïdes ne se limitent pas à la locomotion et il est intéressant de se demander s'il est possible de combiner aux tâches de navigation d'autres tâches asservies par les données capteurs afin de réaliser des comportements complexes. De manière indirecte, la question qui se pose ici est celle de la limite à placer entre d'une part raisonnement numérique et d'autre part raisonnement logique. Ce problème récurrent de la robotique et pour lequel il n'existe pas, de consensus au sein de la communauté trouve ici une solution élégante. En effet, un contrôleur fondé sur le paradigme de la pile de tâches ne se limite pas à une description d'objectifs ou de contraintes robotiques dans des espaces plus naturels que l'espace des configurations ou l'espace cartésien, il ouvre surtout la voie à des mécanismes de supervision décidant à quel moment insérer telle ou telle tâche à un niveau de priorité donné. La pile de tâches réalise donc la jonction entre d'une part, le monde du calcul numérique puisque la finalité du système est de calculer la commande du système et d'autre part le monde de la logique dans lequel les utilisateurs humains souhaitent exprimer leur *desiderata* au système robotique : va jusqu'à la cuisine, apporte-moi la bouteille, ouvre la porte, etc. Il est clair que ce type d'ordre nécessite la résolution d'abstractions et que la logique mathématique semble être un moyen particulièrement adapté pour y parvenir. Ces mécanismes de décision de haut niveau sont appelés "superviseurs" et ont pour objectif d'instancier et d'orchestrer tous les composants d'une architecture robotique. Dans le cadre de notre architecture, il faut donc pouvoir instancier des piles de tâches à partir d'une description de haut niveau des ordres passés au robot. Ce chapitre propose un langage de description des tâches permettant de faire le lien entre représentation logique et représentation numérique.

Nous allons commencer par détailler l'état de l'Art et en particulier les autres travaux ayant trait aux architectures robotiques haut niveau et à l'ordonnancement de tâches ainsi que plus généralement aux applications robotiques complexes. Dans un second temps, le langage de description sera décrit et quelques scénarii types seront démontrés. Enfin, nous nous concentrerons sur les problèmes d'asservissement posés par les robots humanoïdes avant de conclure.

4.2 État de l'Art

La perception et l'exécution de tâches asservies en robotique représentent deux domaines particulièrement actifs. Un livre de référence sur la localisation robotique et le SLAM, le domaine lié à la cartographie et localisation simultanée en robotique est Thrun et collab. (2005). Les travaux de navigation en robotique humanoïdes sont plus rares. On peut citer les travaux de James Kuffner, Joel Chestnutt, Koichi Nishiwaki – 正西脇光一 – Michel et collab. (2005); Ozawa et collab. (2005); Chestnutt et collab. (2003); Nishiwaki et collab. (2002) de l'Université de Freiburg en Allemagne Oßwald et collab. (2010). Dans le domaine la vision pour les humanoïdes, l'utilisation de l'asservissement visuel a été tenté Dune et collab. (2010) ainsi que des techniques de SLAM telles que Stasse et collab. (2006); Kwak et collab. (2009). Une introduction à l'asservisse-

ment visuel est Chaumette et Hutchinson (2006); Chaumette et collab. (2007). La possibilité d'utiliser un robot humanoïde pour la modélisation automatique d'objets 3D a également été envisagée par Foissotte et collab. (2009); Stasse et collab. (2008a). Une approche fondée sur la replanification à partir de données vision sur HRP-2 a également été publiée dans Dang et collab. (2011).

4.3 Description d'un mouvement robotique complexe

Décrire le comportement d'une pile de tâches revient à décrire deux éléments primordiaux : d'une part les tâches exprimées dans le solveur et de l'autre les différents changements d'état de la pile au cours du mouvement. Concernant les tâches, il s'agit ici de représenter des fonctions mathématiques génériques ne présentant pas de point commun permettant une représentation générique de ces dernières. Par exemple si l'on se limitait aux fonctions linéaires $\mathbf{A}(t)\mathbf{x}+\mathbf{b}(t)$, encoder la matrice $\mathbf{A}$ et le vecteur $\mathbf{b}$ serait envisageable pour un ensemble discret de valeurs t données. Le cas évoqué est trop contraint pour notre problème et les modélisations informatiques développées dans le Chapitre 2 n'aident pas : elles ont pour but de venir fournir un modèle pour l'expression d'une fonction mathématique sous la forme d'un algorithme et non pas sous la forme d'une donnée structurée pouvant facilement être encodée. Le choix a donc été fait de définir un ensemble de tâches permettant de réaliser un certain nombre d'actions intéressantes. Rien n'empêche alors cet ensemble de "primitives" d'être étendu au cours du temps mais il nécessite l'extension du modèle de description défini ici.

La seconde partie est la représentation des changements d'état. Un changement d'état du solveur survient quand une tâche est : ajoutée, supprimée ou bien encore quand sa priorité est modifiée. De manière générale les transitions peuvent être, soit temporelles – avance de 5 mètres puis saisis la poignée de la porte –, soit logique – si tu es à moins de 10 cm de la position finale, arrête-toi –. Pour comprendre la stratégie choisie, il faut garder à l'esprit que pour réaliser un scénario complexe certains comportements seront intégrés dans la boucle temps réel tandis que d'autres – typiquement les informations capteurs – seront traitées à l'extérieur. Les deux catégories d'événements, temporelle ou logique, ne mettent pas en jeu les mêmes mécanismes. Les événements temporels sont décidés à l'avance et doivent être exécutés avec une grande précision pour assurer un comportement correct du système tandis que les événements logiques sont le résultat d'un mécanisme de décision externe.

De ce fait, la stratégie adoptée a été de permettre la représentation de transitions temporelles dans le modèle de description uniquement tandis que l'on considère que les transitions événementielles, de fait plus lentes et difficiles à encoder, sont gérées à l'extérieur du contrôleur par un logiciel décisionnel ayant la possibilité de regénérer le mouvement s'il devient invalide suite à la réception d'une nouvelle donnée capteur. De nombreux logiciels adaptés à cette tâche ont été développés tel que SMACH [1].

1. Site web officiel : `http://www.ros.org/wiki/smach`

4.3.1 Primitive de mouvement

La première tâche a été de définir des primitives de mouvement de haut niveau. Les primitives proposées sont :

1. La locomotion : mouvement synchronisé des deux jambes et du centre de masse du robot pour réaliser un déplacement tout en assurant sa stabilité.
2. La manipulation et le contact : mouvement réalisant le placement d'une partie spécifiée du robot à un emplacement donné dans l'espace euclidien. La tâche peut contraindre la position et/ou la rotation du corps à positionner.
3. Regard ou asservissement visuel : le robot doit maintenir un point dans son champ de vision.

4.3.2 Primitive de locomotion

Une tâche de locomotion est définie initialement comme une série de points de contact à réaliser pour atteindre une position finale. Chacun des points de contact étant annoté par l'effecteur devant réaliser le contact à cet endroit. À partir de ces informations, une trajectoire des effecteurs réalisant les appuis est déduite ainsi que du centre de masse pour préserver l'équilibre dynamique du système. Afin de pouvoir utiliser les techniques décrites dans le Chapitre 3, nous nous limiterons à la marche sur un sol plat utilisant le pied gauche et le pied droit du robot. Le calcul de la trajectoire des effecteurs et du centre de masse est encore un problème ouvert à l'heure actuelle. Les modèles simples peuvent être embarqués dans les contrôleurs au prix de nombreuses simplifications détaillées dans le chapitre précédent tandis que les modèles les plus compliqués nécessitent une résolution hors du contrôleur rendant le comportement du système non réactif. Le chapitre précédent a proposé une stratégie pour fusionner les avantages des deux approches. Nous considérons ici la totalité de l'approche développée au cours du chapitre précédent comme une implémentation d'une primitive de locomotion. En particulier, l'erreur de positionnement du robot est considérée comme une entrée de la primitive de mouvement.

Tâche d'alignement de deux repères

Cette primitive de mouvement repose principalement sur la définition d'une tâche où le solveur doit positionner un corps du robot à un endroit précis à la fois en rotation et en translation. C'est cette tâche qui va permettre de faire suivre la trajectoire des pieds notamment.

Soit $\mathbf{M}, \mathbf{M}^* \in \mathrm{SE}(3)^2$ respectivement la position actuelle du corps du robot et la position de référence à atteindre. On peut alors définir l'erreur de cette tâche comme :

$$\mathbf{e} = \mathbf{M}(\mathbf{M}^*)^{-1} \tag{4.2}$$

On remarquera que dans (Équation 4.2), $\mathbf{e}$ est un élément de SE(3). Les éléments de SE(3) peuvent s'exprimer de nombreuses façons différentes : matrice homogène – 16 paramètres –, translation et quaternion – 7 paramètres –, vecteur de rotation – 6 paramètres – , etc. Il faut garder à l'esprit qu'une paramétrisation minimale de SE(3) nécessite six paramètres. De fait, tout représentation utilisant

$$\left(\begin{array}{cccc} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{array}\right) \tag{4.1}$$

FIGURE 4.1 – Structure d'une matrice homogène représentant un élément de SE(3) : les coefficients $R_{i,j}$, $(i,j) \in \{1,\ldots,3\}$ forment la matrice de rotation associé à la transformation et $\{t_x, t_y, t_z\}$ le vecteur de translation. Une matrice de rotation ayant elle-même une forme contrainte : elle doit être une matrice orthogonale de déterminant 1.

FIGURE 4.2 – Représentation d'une rotation par un axe de rotation u et une quantité de rotation θ. Une représentation minimale à trois paramètres est possible en posant : $\theta = |u|$.

un espace de dimension supérieure vient avec un ensemble de contraintes à respecter car tous les éléments de cet espace de dimension supérieur ne peuvent faire partie de SE(3). Par exemple, une matrice homogène a une structure bien particulière tel qu'illustré sur la Figure 4.1.

De fait, les représentations non minimales ne constituent pas une solution acceptable pour représenter notre erreur, car elles provoquent l'insertion de contraintes supplémentaires inutiles dans le problème d'optimisation. Il faut donc choisir pour représenter $\mathbf{e}$ une représentation minimale, dans notre cas par une translation et une rotation autour d'un axe.

En effet, toute rotation peut être représentée par trois paramètres $\theta = (\alpha, \beta, \gamma) \in \mathbb{R}^3$. Le vecteur (α, β, γ) normalisé représentant l'axe autour duquel la rotation s'effectue et la norme du vecteur $|\theta|$ la quantité de rotation réalisée autour de cet axe, voir Figure 4.2.

Cette représentation est minimale, car la translation est représentée par trois paramètres et la rotation par trois paramètres :

$$\mathbf{e} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ \theta_x \\ \theta_y \\ \theta_z \end{pmatrix} \tag{4.3}$$

En considérant les trois premiers éléments du vecteur d'erreur ou bien les trois derniers, on peut restreindre la tâche à positionner le corps respectivement en translation ou en rotation uniquement.

Tâche de position du centre de masse

Le second type de tâche nécessaire pour définir la primitive de locomotion est la tâche de positionnement du centre de masse.

Cette tâche nécessite la connaissance du poids de tous les corps du robot m_i où $i \in \mathcal{B}$. $\mathcal{B}$ l'ensemble des corps du robot. Le centre de masse du robot est alors défini comme le barycentre des centres de masse de chaque corps :

$$\mathbf{x} = \frac{1}{\sum_{i \in \mathcal{B}} m_i} \sum_{i \in \mathcal{B}} m_i \mathbf{x}_i \tag{4.4}$$

$\mathbf{x}$ représentation la position du centre de masse du robot et $\mathbf{x}_i$ la position du centre de masse du corps i.

L'erreur de positionnement du centre de masse est alors simplement :

$$\mathbf{e} = \mathbf{x} - \mathbf{x}^* \tag{4.5}$$

La définition de l'erreur ne posant pas de difficulté ici car $\mathbf{x}$ est un élément de $\mathbb{R}^3$.

Définition de la primitive de locomotion

Une tâche de locomotion est par nature critique : un mauvais suivi de la trajectoire de référence du centre de masse ou bien encore de la trajectoire des pieds aboutit à une perte de l'équilibre du robot. De ce fait, établir une priorité entre ces tâches n'a pas de sens. On préférera donc exprimer ces trois tâches – pied gauche, pied droit et centre de masse – de la façon suivante :

$$\mathbf{e} = \begin{pmatrix} \mathbf{e}_{\text{pied gauche}} \\ \mathbf{e}_{\text{pied droit}} \\ \mathbf{e}_{\text{centre de masse}} \end{pmatrix} \tag{4.6}$$

Les équations formulées dans cette section peuvent enfin être paramétrées par le temps courant t afin de permettre une modification de la valeur de référence notée $\mathbf{M}^*$ ou $\mathbf{x}^*$ selon les cas et permettre le suivi d'une trajectoire plutôt que l'atteinte d'une référence constante.

Comme décrit dans le chapitre précédent, l'erreur des tâches décroît exponentiellement – dans le cas où la référence reste constante –. Un suivi suffisamment précis de la trajectoire est alors réalisable en augmentant suffisamment λ le gain associé à la tâche. Dans la mesure où l'erreur initiale de cette tâche est nulle – le mouvement commence à la position actuelle du robot – et varie de manière continue, la tâche ne génère pas de grandes vitesses articulaires malgré le gain important.

Primitive de manipulation

La primitive de manipulation est une instance directe de la tâche d'alignement de repères présentée dans la section précédente. Elle est utilisée pour placer la main à un endroit donné afin de saisir un objet. Le déplacement d'un bras du robot ne mettant pas en jeu l'équilibre du robot tant qu'elle se déplace peu rapidement et sans perturber la tâche de suivi du centre de masse, on peut simplement corriger l'erreur de positionnement de la main avec une interpolation linéaire pour amener la main à la position désirée.

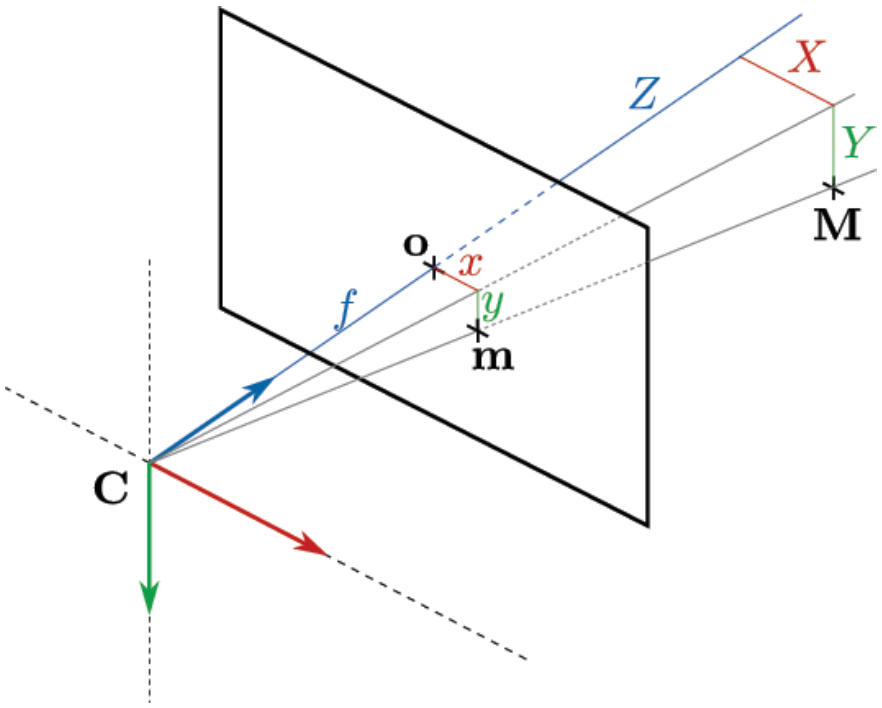

FIGURE 4.3 – Projection du point 3D $M = (X, Y, Z)$ sur le plan image d'une caméra idéale. Les coordonées du point projeté sont ici $m = (x, y)$. f est la distance focale de la caméra et C le centre optique, deux paramètres intrinsèques de la caméra.

Primitive de regard

La dernière primitive introduite ici concerne l'asservissement de la trajectoire du robot afin de préserver des amers dans une zone où elles sont détectables par les capteurs du robot. Sur le robot HRP-2, seules des caméras sont à la disposition des utilisateurs et leur placement dans la tête permet naturellement de définir une tâche de "regard" où l'utilisateur souhaite garder un point au centre de l'image captée par une caméra du robot.

Soit $M = (X, Y, Z) \in \mathbb{R}^3$ un point 3D dont les coordonnées sont définies par rapport à la position de la caméra du robot et $m = (x, y) \in \mathbb{R}^2$ sa projection sur le plan image de la caméra. On a alors la relation suivante :

$$\mathbf{m} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} X/Z \\ Y/Z \end{pmatrix} \tag{4.7}$$

Une fois de plus l'erreur se définit par la soustraction usuelle :

$$\mathbf{e} = \mathbf{m} - \mathbf{m}^* \tag{4.8}$$

4.3.3 Langage de description

Le langage de description adopté est le YAML – Yet Another Markup Language – (Ben-Kiki et collab., 2009). Le fichier est divisé en trois parties :

En-tête fournissant les méta-informations sur le mouvement, notamment sa durée en secondes.

Primitives de mouvement définissant quelles primitives sont jouées et sur quel intervalle, ainsi que la configuration de chaque primitive.

Primitives d'asservissement définissant comment l'erreur de positionnement du robot est estimée au fil du temps.

Primitive de locomotion

Une primitive de locomotion est définie par un ensemble de points de contacts définis sur la forme d'une pile de pas.

Intervalle. Date de début et de fin de la primitive de mouvement.

Empreinte de pas. Liste des pas à effectuer. Les pas sont considérés comme des éléments de SE(2) dans la mesure où l'algorithme de génération de pas fait l'hypothèse d'un sol plan.

Lors du chargement du plan, la génération des trajectoires de pieds et de centre de masse initiales est réalisée hors ligne de manière asynchrone. Tant que le calcul n'est pas terminé, il n'est pas possible de lancer le mouvement.

Primitive de manipulation

Une primitive de manipulation est définie par un corps et une position 6d de référence. L'objectif est de faire coïncider le corps avec la position 6d de référence. Cette tâche peut ne considérer que la rotation ou la translation.

Intervalle. Date de début et de fin de la primitive de manipulation.

Corps à considérer. Nom du corps à considérer. Des noms génériques ont été définis tels que : cheville gauche, cheville droite, poignet gauche, poignet droit, tête, torse et bassin.

Consigne. La position de référence vers laquelle le corps doit être amené. Il peut être de trois types. Soit statique, le corps doit maintenir sa position initiale, soit fixe dans ce cas le corps doit atteindre un point prédéterminé de l'espace, soit dynamique dans ce cas le corps doit suivre un point mobile.

Primitive de regard

La primitive de regard définit comment le robot peut maintenir son regard vers un point 3D, éventuellement mobile.

Intervalle. Date de début et de fin de la primitive de manipulation.

Caméra à considérer. Nom de la caméra utilisée. En pratique, la caméra est définie comme un corps "virtuel" du robot et l'on peut donc potentiellement aligner n'importe quelle partie du corps du robot vers un point donné.

Consigne. La position de référence vers lequel le corps doit pointer. Il peut être de deux types. Soit fixe, dans ce cas le corps doit pointer vers un point prédéterminé de l'espace, soit dynamique et dans ce cas le corps doit pointer vers un point mobile.

Asservissement des primitives sur les données capteur

Une fois la séquence de mouvement définie, il faut encore pouvoir fermer la boucle sur les données capteur. Dans le cadre d'un mouvement complexe, une stratégie intéressante est de pouvoir s'asservir successivement sur plusieurs amers afin de planifier *a priori* quelle est la ou les amers les plus pertinentes à différents instants.

Soit $\mathcal{L}$ un système de localisation. Un système de localisation est défini comme une fonction qui à tout instant fournit une pose estimée d'un corps de référence du robot, dans notre cas le bassin. On a donc :

$$\begin{array}{rccc} \mathcal{L}: & \mathbb{R} & \to & \mathrm{SE}(2) \\ & t & \mapsto & \mathcal{L}(t) \end{array} \tag{4.9}$$

Au cours du mouvement, n systèmes de localisation fournissent une estimation de la pose du robot. Une fois de plus, cette pose est théoriquement dans l'espace 3D SE(3), mais les contraintes physiques font que seuls trois degrés de liberté doivent être réellement estimés : $(x, y, \theta) \in \mathrm{SE}(2)$. Ces trois degrés correspondent à la position 2D de la projection du bassin sur le sol. De ce fait, l'interpolation de n poses 2D et la moyenne des poses à la normalisation de l'angle θ prêt.

On associe à chaque système de localisation une fonction de poids déterminant l'influence relative des systèmes de localisation dans l'estimation finale de la pose :

$$\begin{array}{rccc} m_i: & \mathbb{R} & \to & \mathbb{R} \\ & t & \mapsto & m_i(t) \end{array} \tag{4.10}$$

La fusion des données pour l'estimation de la pose est donc le barycentre des poses 2d :

$$\hat{\mathbf{x}}(t) = \frac{1}{\sum_{i \in n} m_i} \sum_{i \in n} m_i \mathcal{L}_i(t) \tag{4.11}$$

En pratique, chaque système de localisation est représenté de la façon suivante :

Poids. en fonction du temps. Il peut, soit être constant, soit varier dynamiquement.

Erreur. de positionnement en fonction du temps. Elle est fournie par un système de localisation externe.

4.4 Scénarii de mouvements

Nous allons voir ici plusieurs exemples de mouvements décrits en utilisant le langage de description introduit dans la section précédente. L'objectif est à la fois de démontrer la généricité de l'approche par plusieurs scénarii différents ainsi que sa mise en œuvre pratique.

```
duration: 20 # durée complète du mouvement (secondes)

# Éléments de mouvement
motion:
  # Primitive de locomotion
  - walk:
      interval: [0, 20] # Date de début et de fin de la primitive

      # Pile de pas
      footsteps:
      - {x: 0.15, y: -0.19, theta: 0.}
      - {x: 0.15, y: +0.19, theta: 0.1}
      # etc.
```

FIGURE 4.4 – Plan de mouvement pour une séquence de marche non asservie.

4.4.1 Locomotion simple

Le premier scénario consiste en le déplacement du robot le long d'une pile de pas prédéterminée pendant 20 secondes. La description complète du plan est fournie par la Figure 4.4.

La marche est effectuée en boucle ouverte et l'erreur d'exécution n'est pas compensée. Chaque pas est exprimé relativement au pas précédent. Le corps effectuant chaque contact peut être déduit de la séquence d'empreinte de pas : si la translation en y du premier pas dispose d'un signe négatif, le premier pas est effectué avec le pied droit, sinon avec le pied gauche. L'alternance pied gauche/pied droit à chaque pas est ensuite implicite.

Il n'y a pas, à ce niveau de vérification de la faisabilité des pas. C'est le rôle du planificateur de s'assurer préalablement à la génération du plan de mouvement que la séquence est réalisable.

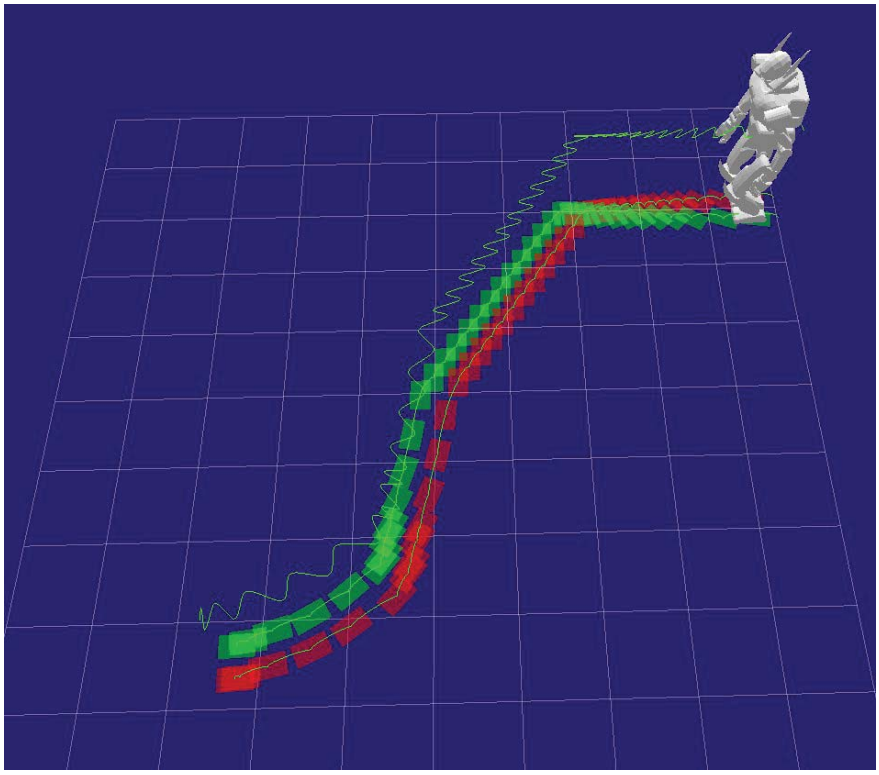

FIGURE 4.5 – Exemple de pile de pas à partir de laquelle une primitive de locomotion peut être calculée.

4.4.2 Locomotion asservie

On peut se demander désormais comment asservir la locomotion en utilisant un composant de localisation. La Figure 4.6 fournit une version alternative du plan précédent ajoutant la correction de pas introduite dans le Chapitre 3.

La seconde partie du plan décrit que l'erreur de positionnement provient d'un unique système de localisation. Dans ce cas, un système de capture de mouvement. Plusieurs systèmes de localisation ont été testés pour fournir une estimation de la position du robot au cours du mouvement :

Capture de mouvement. Ce système permet une localisation quasi-parfaite permettant de s'abstraire de la plupart des problèmes de perception.

Suivi d'un objet. Un système réalisant le suivi d'un objet dans l'image captée par une caméra du robot et reconstituant sa position 3D peut également être utilisé. Dans ce cas, on suppose l'objet fixe dans le monde et on en déduit la position du robot.

SLAM. Les techniques de SLAM – Simultaneous Localization and Mapping – permettent de cartographier l'environnement tout en fournissant une estimation de la position actuelle du robot. La carte enregistrée permet d'assurer une localisation sans dérive quelque soit la longueur du mouvement à condition de pouvoir identifier des amers précédemment détectées régulièrement le long du mouvement.

```
duration: 20 # durée complète du mouvement (secondes)

# correction maximum autorisée sur un pas (x, y en mètre, theta en radians)
maximum-correction-per-step: {x: 0.02, y: 0.02, theta: 0.05}

# Éléments de mouvement
motion:
  # Primitive de locomotion
  - walk:
      interval: [0, 20] # Date de début et de fin de la primitive

      # Pile de pas
      footsteps:
      - {x: 0.15, y: -0.19, theta: 0.}
      - {x: 0.15, y: +0.19, theta: 0.1}
      # etc.

# Asservissement des tâches
servoing:
  # Asservissement via le système de capture de mouvements.
  - mocap:
      weight: 1.
      tracked-body: left-ankle
      perceived-body: left-foot
```

FIGURE 4.6 – Plan de mouvement pour une séquence de marche asservie sur un système de localisation – ici un système de capture de mouvement –.

```
duration: 20 # durée complète du mouvement (secondes)

# correction maximum autorisée sur un pas (x, y en mètre, theta en radians)
maximum-correction-per-step: {x: 0.02, y: 0.02, theta: 0.05}

environment:
  - object:
      name: table
      planned:
        position:
          x: 1.75
          y: 0.3
          z: -0.3
          rx: 0.
          ry: 0.
          rz: 2.

# Éléments de mouvement
motion:
  # Primitive de locomotion
  - walk:
      interval: [0, 20] # Date de début et de fin de la primitive

      # Pile de pas
      footsteps:
      - {x: 0.15, y: -0.19, theta: 0.}
      - {x: 0.15, y: +0.19, theta: 0.1}
      # etc.

# Asservissement des tâches
servoing:
  # Asservissement via le système de capture de mouvements.
  - visp:
      interval: [0, 20] # Date de début et de fin de l'élément d'asservissement.
      weight: 1.
      object-name: table
      position: /tracker_mbt/resultTransform
      camera-velocity: /tracker_mbt/camera_velocity
      frame-name: cameraBottomLeft
```

FIGURE 4.7 – Plan de mouvement pour une séquence de marche asservie sur un système de suivi d'objet.

```
duration: 20 # durée complète du mouvement (secondes)

# correction maximum autorisée sur un pas (x, y en mètre, theta en radians)
maximum-correction-per-step: {x: 0.02, y: 0.02, theta: 0.05}

# Éléments de mouvement
motion:
  # Primitive de locomotion
  - walk:
      interval: [0, 20] # Date de début et de fin de la primitive

      # Pile de pas
      footsteps:
      - {x: 0.15, y: -0.19, theta: 0.}
      - {x: 0.15, y: +0.19, theta: 0.1}
      # etc.

# Asservissement des tâches
servoing:
  # Asservissement via le système de localisation externe (SLAM).
  - control-ros:
      weight: 1.
      topic: /error
      signal: error
```

FIGURE 4.8 – Plan de mouvement pour une séquence de marche asservie sur un algorithme de SLAM. En pratique, dans ce cas l'évaluation de l'erreur est totalement effectuée hors du contrôle.

4.4.3 Scénario d'atteinte avec équilibre quasi statique

Dans ce scénario, le robot place son poignet a un point prédéterminé tout en préservant la position de son centre de masse. Le plan de mouvement est détaillé dans la Figure 4.9.

Contrairement à la tâche de locomotion qui est totalement encapsulée dans une primitive peu paramétrable, le scénario d'atteinte nécessite la définition manuelle des tâches. Quatre tâches sont définies ici : deux pour maintenir les pieds à leur place, une pour maintenir le centre de masse à sa position initiale et une dernière qui génère le mouvement du poignet droit.

Les tâches associées aux pieds et au centre de masse possèdent un gain faible, car l'erreur a une erreur nulle initialement. Inversement, la tâche associée au poignet droit a une erreur maximum initialement et donc une vitesse maximum. Le gain permet alors de contrôler la vitesse de réalisation de la tâche et donc, de manière indirecte, la vitesse de déplacement du bras.

```
duration: 10 # durée complète du mouvement (secondes)

motion:
# Fixe la position des pieds.
  - task:
      # Date de début et de fin de la primitive.
      interval: [0, 10]
      # Type de la tâche: positionnement d'un corps.
      type: feature-point-6d
      # Corps à positionner (cheville gauche).
      operational-point: left-ankle
      # Gain de la tâche.
      gain: 1.
      # Consigne de la tâche: aucun mouvement.
      reference: static
      # Contrainte en translation *et* rotation.
      translation: on
      rotation: on
  - task:
      # Date de début et de fin de la primitive.
      interval: [0, 10]
      # Type de la tâche: positionnement d'un corps.
      type: feature-point-6d
      # Corps à positionner (cheville droite).
      operational-point: right-ankle
      # Gain de la tâche.
      gain: 1.
      # Consigne de la tâche: aucun mouvement.
      reference: static
      # Contrainte en translation *et* rotation.
      translation: on
      rotation: on
# Fixe la position du centre de masse.
  - task:
      # Date de début et de fin de la primitive.
      interval: [0, 10]
      # Type de la tâche: positionnement du centre de masse.
      type: feature-com
      # Gain de la tâche.
      gain: 1.
      # Consigne de la tâche: aucun mouvement.
      reference: static
      # Degré de liberté supplémentaires déverrouillés.
      extra-unlocked-dofs: [18., 19.] # HRP-2 chest dofs.

# Tâche d'atteinte.
  - task:
      # Date de début et de fin de la primitive.
      interval: [0, 10]
      # Type de la tâche: positionnement d'un corps.
      type: feature-point-6d
      # Corps à positionner (poignet droit).
      operational-point: right-wrist
      # Gain de la tâche.
      gain: 7.5
      # Consigne: position finale du poignet droit.
      reference: {x: 0.4, y: -0.3, z: 1.1}
      # Contrainte en translation uniquement.
      translation: on
      rotation: off
```

FIGURE 4.9 – Plan de mouvement pour une tâche d'atteinte.

4.4.4 Scénario d'asservissement visuel de la tête

La dernière primitive de mouvement considérée ici permet d'asservir le regard du robot sur la position d'un point 3D, soit fixe, soit mobile et estimé par un système externe. Le plan correspondant est la Figure 4.10.

```
duration: 25 # durée complète du mouvement (secondes)
# correction maximum tous les deux pas
maximum-correction-per-step: {x: 0.04, y: 0.04, theta: 0.1}

environment:
  - object:
      name: table
      planned:
        position:
          x: 1.75
          y: 0.3
          z: -0.3
          rx: 0.
          ry: 0.
          rz: 2.

motion:
  # Primitive de locomotion
  - walk:
      interval: [0, 20] # Date de début et de fin de la primitive

      # Pile de pas
      footsteps:
      - {x: 0.15, y: -0.19, theta: 0.}
      - {x: 0.15, y: +0.19, theta: 0.1}
      # etc.

  # Asservissement de la tête
  - visual-point:
      interval: [0, 25]
      gain: 1.
      object-name: table
      frame-name: cameraBottomLeft

servoing:
  - visp:
      weight: 1.
      object-name: table
      position: /tracker_mbt/resultTransform
      frame-name: cameraBottomLeft
```

FIGURE 4.10 – Plan de mouvement pour une tâche de marche avec asservissement de la tête.

4.5 De la difficulté à localiser un robot humanoïde

Plusieurs séries d'expérimentation ont été réalisées sur le robot HRP-2 afin de tester le comportement des algorithmes de localisation utilisant des capteurs embarqués.

Le scénario proposé est le suivant : HRP-2 contourne un obstacle, passe entre deux objets, et va poser une balle sur une étagère. La longueur de la trajectoire de marche – 2 mètres environ – et le nombre de pas génèrant une dérive suffisante pour qu'une exécution de ce scénario échoue en boucle ouverte. En effet, passer son bras dans une étagère est un mouvement contraint particulièrement sensible à la position d'arrivée du robot. Une petite erreur dans le positionnement du robot à la fin de la trajectoire engendrera une grande erreur dans la position finale du poignet. La Figure 4.11 illustre le scénario. L'objectif ici est d'utiliser les caméras embarquées pour construire une carte de l'environnement puis pour localiser le robot. La stratégie adoptée ici n'est pas du SLAM – Simultaneous Localization and Mapping – dans la mesure où les images permettant de construire la carte sont traitées hors ligne. On réalise donc un premier mouvement sans correction afin d'acquérir une séquence d'images permettant de construire la carte puis on utilise cette carte pour localiser le robot durant les expérimentations suivantes. La qualité du résultat est ensuite validée en utilisant un système de capture de mouvement.

Durant les expériences, le module de localisation a fourni une estimation de la position du robot à 16Hz. Les images ont une résolution de 320x240 pixels. L'ordinateur sur lequel fonctionne le composant de localisation est un Intel® Core^{TM}2 CPU T7200 @ 2.00GHz avec 2 Gb. de RAM.

4.5.1 Analyse de la précision du système de localisation

La précision finale du système est illustrée par la Figure 4.14. On peut y observer une erreur à la fin de la trajectoire d'environ 0.2 mètres en translation. Plusieurs raisons peuvent expliquer ces mauvais résultats :

1. Carte non métrique et absence de fermeture de boucle,
2. Passage dans des zones où la carte est peu dense ou dans des zones ne possédant que peu d'information visuelle,
3. Flexibilité du robot et/ou mauvaise identification de certains paramètres,

Le premier point pouvant amener à une mauvaise estimation de la localisation du robot est l'absence de fermeture de boucle. Les techniques de localisation utilisant les informations visuelles peuvent être divisées en deux catégories : odométrie visuelle et localisation "globale". L'odométrie visuelle observe les changements entre deux images aux temps t_{n-1} et t_n afin d'estimer la vitesse de la caméra entre les deux images. Cette vitesse est ensuité intégrée pour calculer le mouvement de la caméra. Par conséquent, il se produit une dérive temporelle qui empêche toute tentative de localisation sur le long terme en utilisant cette technique. Inversement, les algorithmes utilisant une carte peuvent identifier les amers déjà détectés auparavant et stabiliser le système. Une limite de ce mouvement est qu'il n'est pas facile pour le système de reconnaître des amers déjà identifiés. Il s'en suit une dérive dans l'estimation de la position du robot.

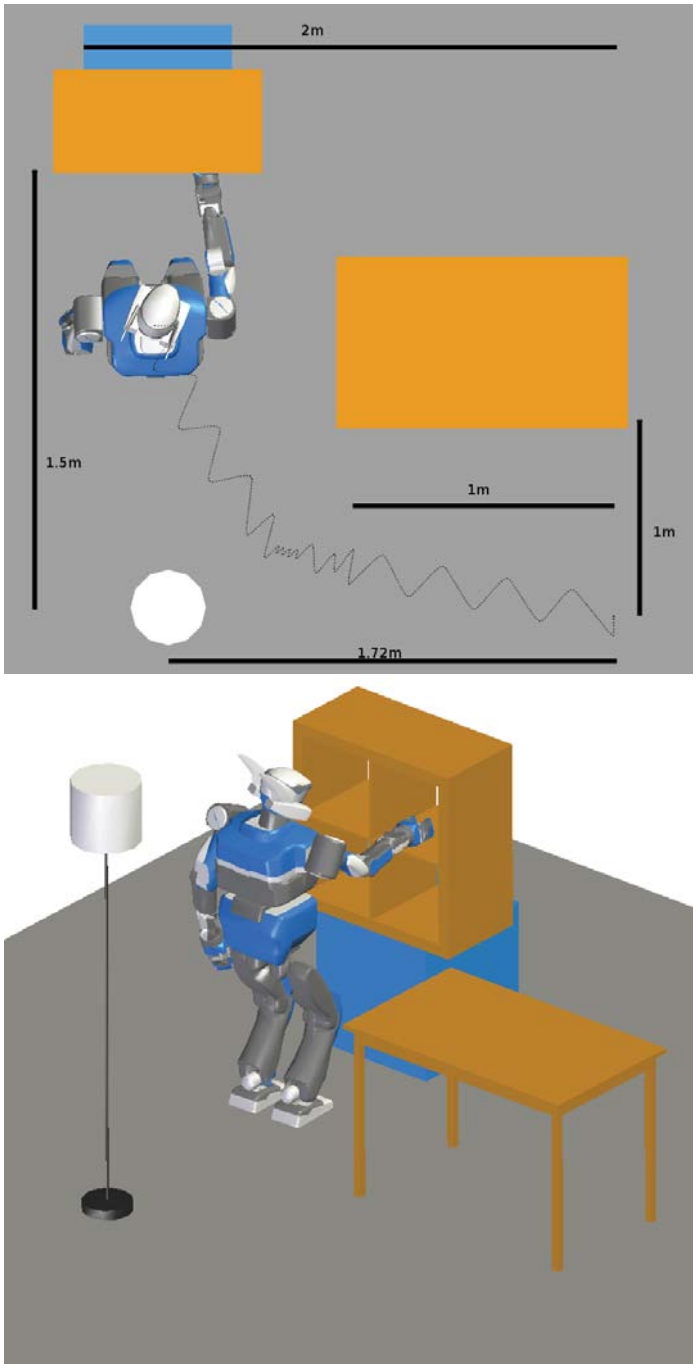

FIGURE 4.11 – Description de l'expérimentation : le robot part de la droite de l'image et progresse vers l'étagère située à gauche pour y déposer une balle.

Le second problème concerne l'environnement en lui-même : à la fin du mouvement, le robot se trouve face au meuble ce qui rend les images captées par les caméras vides de tout amer permettant d'effectuer correctement la localisation. Ce problème ne peut expliquer que partiellement la mauvaise estimation dans la mesure où l'occlusion des caméras n'intervient qu'à l'extrême fin du mouvement.

Le dernier point pouvant amener à une mauvaise estimation de l'erreur est lié à la mauvaise modélisation de la chaîne cinématique du robot. Une première erreur est liée à la présence, au niveau des chevilles du robot, d'une flexibilité passive pouvant être modélisé sous la forme de trois ressorts : deux autorisant un mouvement en rotation et un troisième un déplacement en translation, c'est à dire autorisant une compression de la cheville lors de la marche. Ce dispositif a été conçu pour absorber les chocs et protéger le capteur de force situé dans la cheville mais insère trois degrés de liberté non modélisés et surtout non mesurables, car dépourvu d'encodeur. Cette déformation au niveau de la cheville est donc ignorée dans les calculs ce qui amène à un biais sur la position relative du pied et de la tête du robot. Ce biais est d'autant plus critique dans ce cas que c'est la position des points de contact que l'on essaie de corriger. Une seconde source d'erreur est la calibration de la position de la caméra dans le robot. Les imprécisions de la conception du robot, du système de fixation des caméras rendent nécessaire le calcul de la position relative de la caméra par rapport au corps auquel il est fixé, dans notre cas la tête. Un procédé de calibration des paramètres extrinsèques a donc été mis en place pour estimer ces paramètres. Ces derniers ont été validés en reprojetant le modèle du robot dans l'image comme illustré par la Figure 4.15.

4.5.2 Résultats de la localisation utilisant la vision sur le robot humanoïde HRP-2

L'ensemble de ces facteurs rend la réalisation d'un algorithme de localisation embarqué sur HRP-2 délicat. Pour réaliser une tâche de manipulation, une précision de l'ordre du centimètre est souvent nécessaire. Or, les mauvaises performances du système de localisation engendrent un bruit dans le positionnement du robot qui peut rendre l'exécution de tâches précises délicates. Dans le scénario choisi, la tâche peut être réalisée avec succès pour certains essais, mais le taux d'échec restant très fort, cette méthode ne peut pas constituer une boîte noire permettant de manière automatique de réaliser un mouvement asservi. Une façon de résoudre le problème serait de corriger la position du bras. En effet, entre la caméra et le bras, la chaîne cinématique est bien identifiée. On voit donc ici que l'approche consistant à avoir plusieurs systèmes de localisation simultanément se justifie en pratique. Un composant de localisation "global" tel que celui utilisé ici permet de rendre la navigation robuste sans toutefois être suffisant pour les tâches de manipulation. Pour ces dernières, d'autres stratégies doivent être envisagées comme la détection dans l'image de l'objet à attraper pour asservir la position de la pince. On voit ici que plus qu'une localisation parfaite sur le long terme, il est plus critique pour nos scénarii d'avoir des algorithmes de localisation locaux assurant une erreur minimale à la fin de la réalisation de la tâche. Les algorithmes réalisant une localisation sur le long terme étant plutôt destinés à alimenter les algorithmes de planification que les algorithmes de contrôle nécessitant une haute précision sur la pose estimée du

robot.

4.6 Conclusion

Pour continuer à explorer l'expression de plans de mouvement, il faudrait donc intégrer de nouveaux algorithmes de localisation dédiés à l'asservissement d'une tâche spécifique plutôt que d'envisager le problème comme un bloc unitaire. La seconde partie des travaux reste d'intégrer directement dans la phase de planification une étape de planification des asservissements. L'idée est de planifier le mouvement tel que le robot privilégiera le passage dans des zones où la qualité de la localisation est maximale. En particulier, la trajectoire de la tête sera calculée afin de laisser les amers détectables le plus longtemps possible utilisables. Ce planificateur pourra alors générer la partie asservissement du plan de mouvement automatiquement. L'intégration de ces deux éléments reste un objectif clé afin de pouvoir asservir n'importe quel mouvement sur un robot humanoïde de manière automatique.

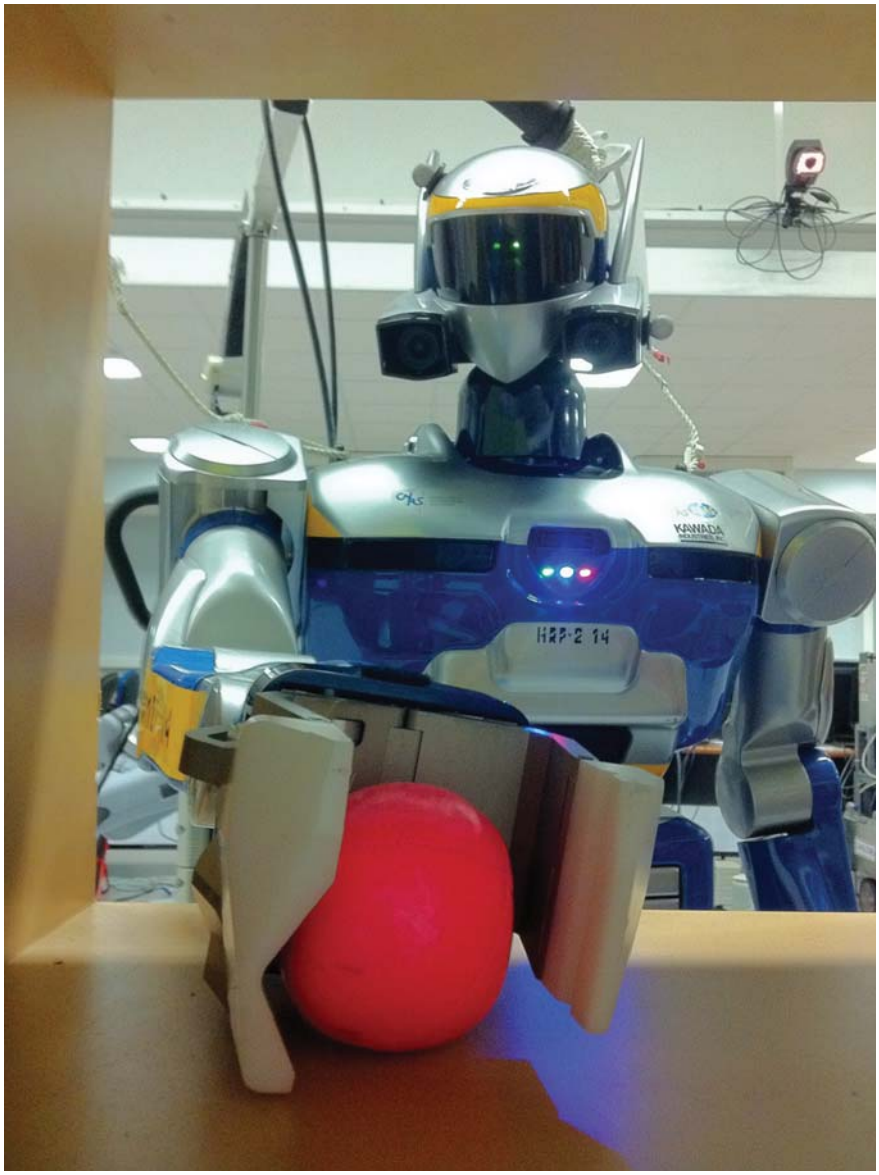

FIGURE 4.12 – HRP-2 place une balle dans une étagère (I).

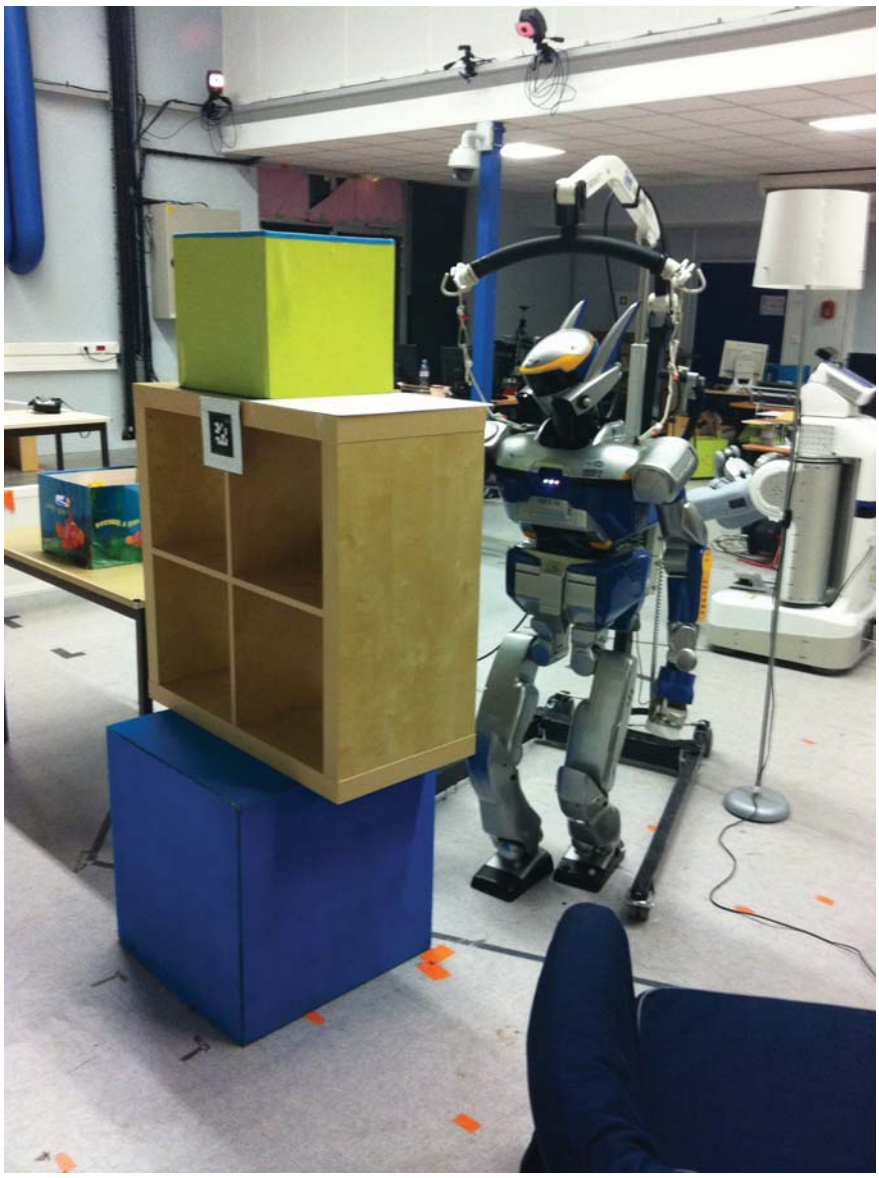

FIGURE 4.13 – HRP-2 place une balle dans une étagère (II).

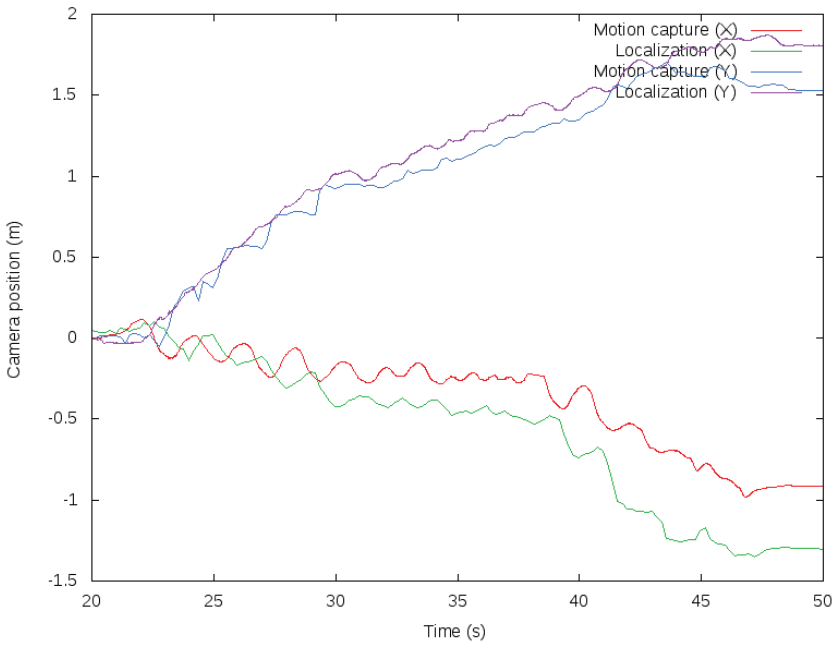

FIGURE 4.14 – Précision de l'algorithme de SLAM par rapport au système de capture de mouvemement.

FIGURE 4.15 – Reprojection du modèle du robot dans l'image captée par une caméra embarquée pour valider la calibration des paramètres de cette dernière.

5 Architecture robotique

> The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.
>
> Niklaus Wirth

Ce chapitre est dédié à la conception d'architectures robotiques complexes permettant la validation de nouveaux algorithmes robotiques. Les précédents chapitres ont montré une progression d'une approche qui est dédiée purement à la génération de trajectoires par l'utilisation d'outils d'optimisation numérique vers l'exécution de scénarii dont la complexité a augmentée incrémentalement. Ce passage de la génération de trajectoire sans "conscience" de ce qu'est un système évoluant dans le monde réel vers une application robotique réelle ne va pas sans la nécessité d'intégrer et de maîtriser une grande variété d'algorithmes et de technologies tout en les utilisant conjointement afin de finalement, atteindre un objectif de haut niveau. Ce chapitre va donc décrire l'architecture qui a été mise en place sur le robot humanoïde HRP-2 afin de réaliser un ensemble de scénario expérimentaux, certains faisant partie intégrante des chapitres précédents et d'autres étant le résultat de travaux de recherche disjoints, mais prenant appui sur l'architecture développée ici. Ce chapitre tente également de fournir des conseils et une méthodologie pour la conception d'applications robotiques.

5.1 Architecture

Les systèmes robotiques peuvent être décomposés en trois grandes parties :

1. La couche de perception qui analyse l'environnement autour du robot et construit un modèle du monde,
2. La couche de décision qui va tenter de résoudre la tâche donnée au système tout en prenant en compte l'état actuel du monde,
3. La couche action qui va utiliser les capacités du robot pour réaliser la tâche proprement dite. En particulier, les capacités d'actionnement du robot sont ici utilisées pour impacter le monde environnant.

Cette architecture largement décrite par la littérature est encore d'actualité pour les robots humanoïdes. En effet, cette catégorisation est issue de la présence dans un système robotique de boucles lentes et de boucles rapides. Typiquement, la prise de décision est un processus lent – qui peut prendre plusieurs secondes –, la planification de mouvements est un exemple de ce type de processus. Au contraire, la couche action contrôlant les actionneurs nécessite le plus souvent d'être extrêmement réactive et impose des contraintes importantes en terme de temps réel. Il faut pouvoir garantir que la boucle de contrôle peut être évaluée à une fréquence donnée ce qui contraint à la fois le type d'algorithmes pouvant être exécuté dans cette boucle ainsi que le volume de données pouvant être traité. Enfin, la couche de perception est, elle, en général plus lente que le contrôle et dépendante de la vitesse à laquelle les capteurs du robot peuvent fonctionner. Qui plus est, il arrive parfois que les capteurs acquièrent des données plus vite qu'elles sont traitées et certaines données sont alors ignorées silencieusement. On a donc ici trois types de comportements extrêmement différents.

Les couches perception et décision nécessitent une grande puissance de calcul pour pouvoir assurer une réactivité suffisante. Cette puissance de calcul est parfois déportée sur un ordinateur spécifique. Par exemple dans le cadre du robot HRP-2, deux ordinateurs sont embarqués. Le premier est dédié à la boucle de contrôle temps réel et est relié aux actionneurs tandis que le second est relié

aux capteurs – ici les caméras embarquées – et supporte les algorithmes de décision et de perception.

Afin de découpler les algorithmes robotiques, chaque algorithme est implémenté au sein d'un composant robotique. En pratique, un composant robotique est un processus – c'est-à-dire un programme indépendant – pouvant communiquer avec un ou plusieurs autres composants. La distribution des calculs sur plusieurs ordinateurs rend nécessaire la définition d'un protocole de transmission des données afin de pouvoir assurer une bonne interprétation de ces dernières au sein d'un environnement informatique hétérogène. Un exemple est la représentation des nombres au sein des architectures informatiques : une architecture peut être dit "big endian" ou "little endian" selon l'ordre dans lequel les bits représentants le nombre sont enregistrés. Dans une architecture "big endian" les bits ayant les poids les plus forts sont contenus en mémoire en premier tandis que les architectures "little endian" adoptent un ordre inverse. Ce type de problème rend nécessaire la définition d'un modèle de communication entre composants.

Les architectures robotiques autorisent généralement la communication entre deux composants, ou nœuds, soit sous une forme discrète, soit sous une forme continue. La forme discrète est appelée "service" et se modélise sous la forme d'un appel de fonction distant. Un algorithme est généralement divisé en fonctions, ces dernières formant des blocs logiques pouvant être combinés entre eux pour réaliser des comportements de plus haut niveau. Les services fournissent un moyen d'appeler une fonction qui sera non pas exécutée au sein du processus courant, mais dans un autre processus, voire sur un autre ordinateur de manière transparente. La difficulté résidant dans le passage des arguments et la transmission du résultat. Il est nécessaire que cette fonction soit définie selon des règles particulières afin de s'asssurer que les données peuvent être correctement transmises via le réseau sans compromettre leur intégrité. Cette phase, dite de "sérialisation", assure le fonctionnement de ces mécanismes dans des environnements informatiques hétérogènes. La seconde forme de communication sert à modéliser des flux de données continus et est appelée "topic". Dans ce cas, le premier composant communique à un ou plusieurs autres des données mises à jour régulièrement. De la même façon, un processus de sérialisation est nécessaire pour assurer que les données peuvent être transmises même si un ou plusieurs autres composants ne font pas partie du même processus ou ne sont pas exécutés sur le même ordinateur.

De nombreux outils logiciels implémentent ces mécanismes sous différentes formes. Le choix réalisé sur le robot humanoïde HRP-2 a été d'utiliser ROS – Robotics Operating System –, un projet communautaire initié par la société californienne Willow Garage. Le mécanisme de communication ainsi qu'une grande partie de la modélisation du processus de perception se fonde sur les outils développés dans le cadre de ce projet. Les sections suivantes vont détailler les différents composants fonctionnant sur le robot humanoïde HRP-2.

5.1.1 Contrôle

Sous le terme de "contrôle" est désigné le composant robotique chargé de calculer la commande qui sera envoyée aux actionneurs. Il est impératif que cet ordre déterminant le prochain état que les servomoteurs du système vont s'efforcer d'atteindre soit envoyé à une fréquence fixe. Sur le robot humanoïde HRP-2, la fréquence de la boucle de contrôle est de 200Hz, il faut donc que

la boucle de contrôle ne mette pas plus de 5ms pour être évaluée. Les noyaux des systèmes d'exploitation multitâches ne fournissent pas de garantie forte sur la réactivité des logiciels exécutés. Il est donc nécessaire d'utiliser des noyaux dédiés afin d'assurer que la boucle de contrôle soit exécutée en permanence à la fréquence voulue. Ces contraintes impliquent que les composants de contrôle se doivent d'être aussi minimaux que possible et sont souvent monolithiques. À ce sujet, les contrôleurs robotiques partagent une grande ressemblance avec les noyaux des systèmes d'exploitation. Ils assurent également la sûreté du système : une mauvaise commande envoyée aux moteurs peut, très facilement, entraîner la destruction des actionneurs.

Dans le cas de systèmes robotiques complexes, il est courant d'avoir plusieurs contrôleurs exécutés successivement au sein de la boucle de contrôle. Ainsi l'architecture d'HRP-2 se fonde sur un système de plug-in permettant de charger des contrôleurs "à chaud". Chaque contrôleur est alors modélisé par une machine à état très simple illustrée par la Figure 5.1. Le contrôleur passe alors par les états suivants :

- Initialisation sans contrainte de temps réel.
- Initialisation avec contrainte de temps réel.
- Exécution.
- Destruction avec contrainte de temps réel.
- Destruction sans contrainte de temps réel.

Le passage d'un état à l'autre se réalisant dans l'ordre, avec un bouclage sur la phase d'exécution autant qu'il soit nécessaire.

L'architecture de contrôle est divisée en trois contrôleurs :

Pile de tâches. À partir d'un ensemble de tâches, la résolution de la pile de tâches est calculée dans la boucle de contrôle afin de calculer la commande à envoyer aux moteurs. HRP-2 étant commandé en position, la commande envoyée correspond à la position articulaire des différents articulations du système.

Stabilisateur. HRP-2 intègre un mécanisme d'absorption passif des chocs appelé "silent blocs". Ce système passif intègre au niveau des chevilles une compliance difficile à modéliser et à prendre en compte dans le calcul de la commande. La centrale inertielle du robot permettant d'estimer l'attitude du bassin, le stabilisateur peut déduire l'état des flexibilités des chevilles et, à l'aide des capteurs de force placés dans les chevilles, réaliser un asservissement bas-niveau sur la trajectoire du ZMP désiré. Ce mécanisme permet de compenser la modélisation sommaire de la dynamique réalisée, par exemple, par l'utilisation du modèle simplifié du ZMP ainsi que des éventuelles erreurs de modélisation de la structure du robot. Ce système compense les erreurs de suivi du ZMP en modifiant les valeurs des articulations des jambes du robot.

Communication. Le dernier élément communique vers l'extérieur les informations liées au contrôle du robot tel que les commandes envoyées, réellement exécutées – c'est-à-dire après stabilisation –, ainsi que les sorties des capteurs tels que la centrale inertielle et les capteurs de force placés aux chevilles et poignets du robot.

On notera que pour le dernier contrôleur, la communication réseau étant implémentée en TCP/IP, il n'est pas possible de garantir le temps pris par l'envoi des données et il est donc impossible d'envoyer des données en TCP/IP sans

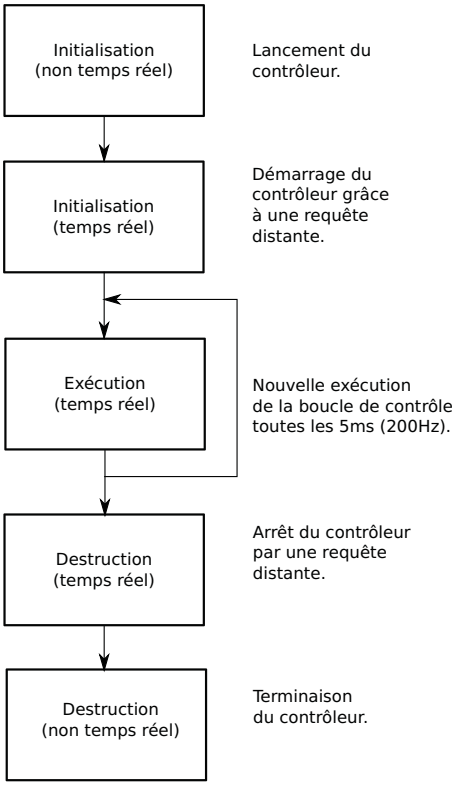

FIGURE 5.1 – Schéma de fonctionnement d'un contrôleur temps réel. Plusieurs contrôleurs de ce type peuvent être lancés. Dans ce cas, chaque tour de la boucle de contrôle correspond à une exécution de la boucle de contrôle de chaque contrôleur actif – initialisé – dans l'ordre dans lequel ils ont été créés.

compromettre la boucle temps réel. Cette partie est donc divisée en plusieurs fils d'exécution non bloquants afin de garantir la sûreté du système. En pratique, chaque "topic" permettant la communication de données vers l'extérieur publie les données dans un fil d'exécution séparé et non temps réel. Le fil d'exécution temps réel lui tente de copier la donnée vers une variable lue par ce fil d'exécution. Si ce dernier est en train d'y accéder et qu'elle n'est pas accessible, un verrou protège l'accès à la ressource et la publication de la donnée pour ce tour de boucle est ignorée. En pratique, la communication vers l'extérieur est réalisée à 20Hz. En moyenne, une donnée sur dix est donc envoyée afin d'éviter d'engorger le système.

En terme de conception, les composants nécessitant un rafraichissement à une vitesse supérieure doivent être intégrés à la boucle de contrôle. Pour les autres données, il faut pouvoir, via le schéma de calcul, utiliser les informations temporelles attachées aux données capteur envoyées pour gérer de manière consistante le retard inhérent à toute donnée provenant d'une architecture distribuée.

5.1.2 Vision

La vision est l'élément central de la perception sur un robot humanoïde. Elle est caractérisée par une particularité qui détermine tous les traitements qui lui sont affectés : les caméras produisent des quantités de données extrêmement importantes. De ce fait, il est important d'éviter de réaliser des traitements inutiles et il faut également éviter au maximum les copies. Le traitement des images est souvent réalisé en série : on acquiert une image, on la convertit, on la rectifie et puis on cherche à en tirer des informations de plus haut niveau. Ces traitements en série se modélisent particulièrement bien sous la forme d'un tuyau dans lequel les images passent les unes après les autres. La stratégie classique de la robotique tend alors à dédier un composant par traitement afin de garder un système modulaire, mais la taille des données rend cela difficile. En effet, si chaque composant vit dans un processus différent, il est nécessaire de passer par des mécanismes de mémoire partagée afin d'éviter les copies en mémoire. ROS a choisi une seconde stratégie en offrant la possibilité d'agglomérer plusieurs composants dans un seul processus [1]. La sérialisation devient donc totalement inutile au prix d'une plus grande complexité dans l'écriture des nœuds et d'une forme d'insécurité : si un problème advient dans n'importe lequel de ces composants, il provoquera l'arrêt de la totalité du système. Cependant, les traitements étant effectués en série, devoir relancer tous les traitements en cas de panne n'amène pas de perte significative de qualité de service.

Nous allons donc désormais nous attacher à développer les différents composants de vision qui ont été utilisés sur le robot HRP-2 et agglomérés dans la chaîne de traitement des images du robot.

Acquisition des images L'acquisition des images est réalisée par un composant communiquant avec le pilote FireWire. Les quatre caméras du robot sont des Flea2 du fabricant PointGrey. Ces caméras sont reliées à un FireWire IEEE 1394b. Les limitations du bus ne permettent pas l'acquisition des données sur les quatre caméras simultanément, on peut donc passer de la paire de caméras

1. L'agrégation de plusieurs nœuds se nomme "nodelet" : `http://ros.org/wiki/nodelet`.

grand-angle à la seconde paire de caméras. La première paire est dédiée à la perception de l'environnement au détriment de la précision tandis que la seconde dispose d'un champ plus resserré, adapté à la manipulation fine d'objets notamment.

Les images acquises utilisent un encodage de Bayer. Cet encodage, nommé du nom de son inventeur, Bryce E. Bayer chercheur au sein de la société Kodak, réalise l'encodage d'images couleurs sur 8 bits au lieu des 24 bits habituellement nécessaires pour représenter les trois canaux utilisés pour encoder les trois couleurs primaires : rouge, vert et bleu. Cet encodage encode la couleur verte sur 50% du volume de données totale tandis que bleu et rouge représentent 25% chacun. Cette prépondérance du vert est fondée sur la plus grande sensibilité au vert des cônes, les photorécepteurs présents au fond de l'œil.

Traitements des images Une fois les images acquises, il est important de les convertir dans un espace colorimétrique adapté à leur traitement. La première étape est donc de dématricer les images encodées en Bayer vers des images monochromatiques ou couleurs 24 bits dans l'espace RGB habituel. Dès maintenant, un branchement s'effectue entre les traitements utilisant les images couleur et les traitements utilisant les informations de luminance uniquement. Afin de fournir une expérience unifiée, les deux "topics" sont créés, mais les traitement ne sont effectués qu'à partir du moment où au moins un client se connecte à ce dernier. On peut donc offrir une large palette de possibilités sans mettre à mal les performances du système. Le dématriçage des images encodées en Bayer reconstruit les couleurs manquantes en interpolant grâce à la couleur des pixels adjacents. Plusieurs techniques ayant des coûts de calcul différents existent selon les cas : interpolation bilinéaire, interpolation bicubique ou bien encore rééchantillonage de Lanczos.

La seconde étape consiste à rectifier les images. Nous nous intéresserons d'abord au cas de la rectification monoculaire. Dans ce cas, l'objectif de ce procédé est de se ramener au cas de la caméra parfaite dit "pinhole" – ou sténopé en français – où la projection du point 3D P sur le plan image donne un point 2d p qui correspond parfaitement à la projection de ce dernier en passant par le point focal de la caméra C : $p_x = P_x/P_z$ et $p_y = P_y/P_z$ où P_x, P_y, P_z sont les coordonnées du point dans l'espace 3D et p_x, p_y les coordonnées 2D du point p dans le plan image. Ce n'est pas jamais totalement le cas en réalité : le plan image n'est pas un plan parfait, l'axe optique n'est jamais exactement au centre de la caméra et les objectifs réalisent une distorsion des rayons lumineux captés. Pour pouvoir raisonner sur ce modèle idéal, on va donc traiter l'image afin qu'elle puisse être considérée comme provenant d'un capteur idéal ayant le comportement d'un sténopé, à la différence de la caméra réelle. Il faut pour cela calibrer la caméra et identifier ses paramètres intrinsèques : (u_0, v_0) la position de l'axe optique dans l'image, (p_x, p_y) la taille des pixels permettant de réaliser la conversion des pixels vers les mètres et un ensemble de paramètres permettent d'identifier la distorsion de l'image originale. Plusieurs modèles sont décrits par la littérature, mais dans notre cas une estimation de la distorsion radiale est suffisante pour compenser l'utilisation d'un objectif grand-angle.

Le processus de calibration est réalisé en utilisant une mire que l'on place à différents endroits relativement à la caméra. Connaissant la taille des points et leur écartement, on peut alors, par la résolution d'un problème type moindres

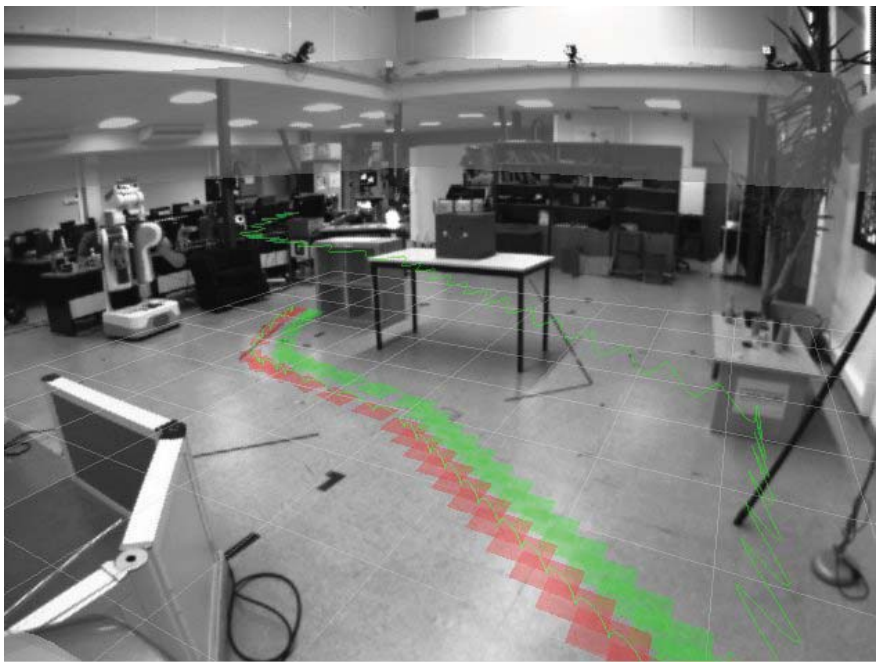

FIGURE 5.2 – Affichage de la pile de pas que le robot va réaliser en réalité augmentée.

carrés linéaire, réaliser une estimation des paramètres intrinsèques des caméras du système.

Une fois de plus, ce traitement est disponible dans deux variantes différentes : monochromatique et couleur.

Vision stéréoscopique Le dernier traitement concernait la vision monoculaire. Il est courant, en robotique, d'avoir recours à la vision stéréoscopique afin de pouvoir reconstruire la profondeur d'un objet visible dans les deux caméras. Il est alors nécessaire d'estimer la transformation relative entre les deux caméras afin de pouvoir rectifier l'image de la seconde image de la paire de caméras stéréo. Une fois ce processus effectué, on peut estimer que les images de deux caméras ont été prises par des capteurs virtuels dont les plans images sont coplanaires. On peut alors faire en sorte que la projection d'un point 3D se fasse sur la même ligne tant sur la caméra de gauche que sur la caméra de droite. Le problème de l'appariement des points se ramène alors à un problème unidimensionnel de complexité moindre permettant la construction d'une carte de profondeur. Cette carte est disponible sous la forme d'un topic, mais les données de profondeur sont également disponibles sous la forme d'un nuage de points. Dans ce dernier, on associe à chaque point la couleur du pixel correspondant dans l'image. Ce topic est particulièrement utile lorsque l'on souhaite utiliser des algorithmes de traitements de nuages de points 3D tels que PCL – la Point Cloud Library –.

Une fois de plus, la totalité du processus de vision peut fonctionner dans un

processus unique afin d'éviter les copies. Cette architecture, également adoptée sur le robot PR2 de Willow Garage, fournit donc un compromis intéressant entre rapidité, souplesse d'utilisation et modularité.

5.1.3 Capture de mouvement

Une alternative à la vision embarquée est l'utilisation des systèmes de capture de mouvement. Ces derniers, à l'origine destinés à étudier les mouvements humains, ont été largement détournés afin de pouvoir suivre les mouvements d'objets autour du robot et s'abstraire des difficultés propres à la perception en robotique.

Un composant réalisant l'interface entre d'une part le système de capture de mouvement et d'autre part l'architecture robotique a été écrit. Ce dernier permet de suivre les mouvements de plusieurs objets simultanément et publie une estimation de leur pose via le système de "topics" précédemment décrit.

5.1.4 Diagnostics et sûreté

L'utilisation d'un robot humanoïde de grande taille présente toujours un risque, et ce dernier tend à croître quand l'architecture devient plus complexe et intègre des composants moins fiables. Pour ce faire, il est donc nécessaire de pouvoir obtenir un état complet du système de manière unifiée. Ce dernier est disponible sur le robot et est illustré par la Figure 5.7. De plus, il est nécessaire à tout moment de pouvoir avoir un retour si le mouvement exécuté présente le risque d'endommager le robot. Ces risques se divisent en différentes catégories :

Endommagement des capteurs. Les capteurs de force sont conçus pour résister à un impact de 1000N au plus. Il est recommandé de ne pas aller au-delà de 800N par mesure de sécurité. Les autres capteurs ne présentent pas de risque particulier.

Endommagement des actionneurs. Les actionneurs peuvent fournir un couple maximal. Au-delà de ce dernier, il risque de chauffer de manière trop importante et d'être rendu inopérant. On peut donc fixer une borne "raisonnable" qui ne devrait jamais être dépassée. Évidemment, ceci ne représente qu'une approximation très simple des capacités maximums des actionneurs. En particulier, un retour sur la température des actionneurs se révélerait être un indicateur intéressant, mais n'est malheureusement pas disponible en l'état actuel des capacités du robot.

Instabilité lors de la locomotion. Il est possible d'estimer la position du ZMP à partir des forces mesurées par les capteurs de force. Cette estimation permet d'avertir l'utilisateur quand le ZMP se rapproche dangereusement de la limite du polygone de sustentation.

Déviation des horloges des ordinateurs embarqués. Les ordinateurs embarqués synchronisent leurs horloges en utilisant le protocole NTP – Network Time Protocol –. L'ordinateur dédié à la décision et la perception se synchronisent sur un serveur de temps distant. L'utilisation d'un lien WiFi et la présence de routeurs intermédiaires rendent la synchronisation relativement peu précise. Cependant, le problème n'est pas très gênant, car le véritable enjeu reste de faire en sorte que les deux PCs embarqués gardent leurs deux horloges synchronisées. Le lien réseau étant direct, la

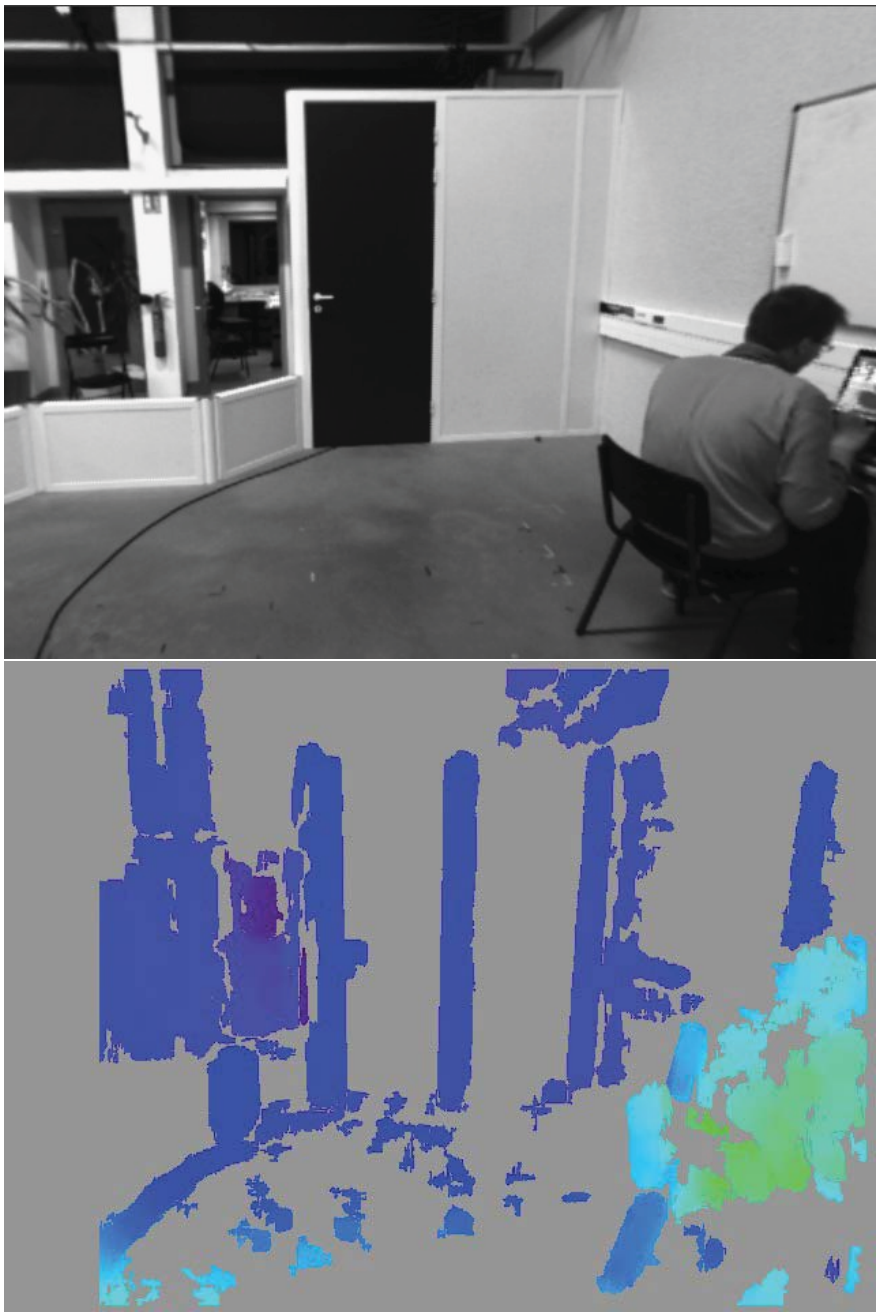

FIGURE 5.3 – Carte de profondeur calculée en temps réel. Une couleur claire correspond à une faible profondeur, une couleur foncée à une profondeur importante. Les zones grises indiquent les endroits où aucune correspondance n'a pu être établie entre les images de deux caméras. Elles correspondent typiquement à une zone peu texturée comme un mur.

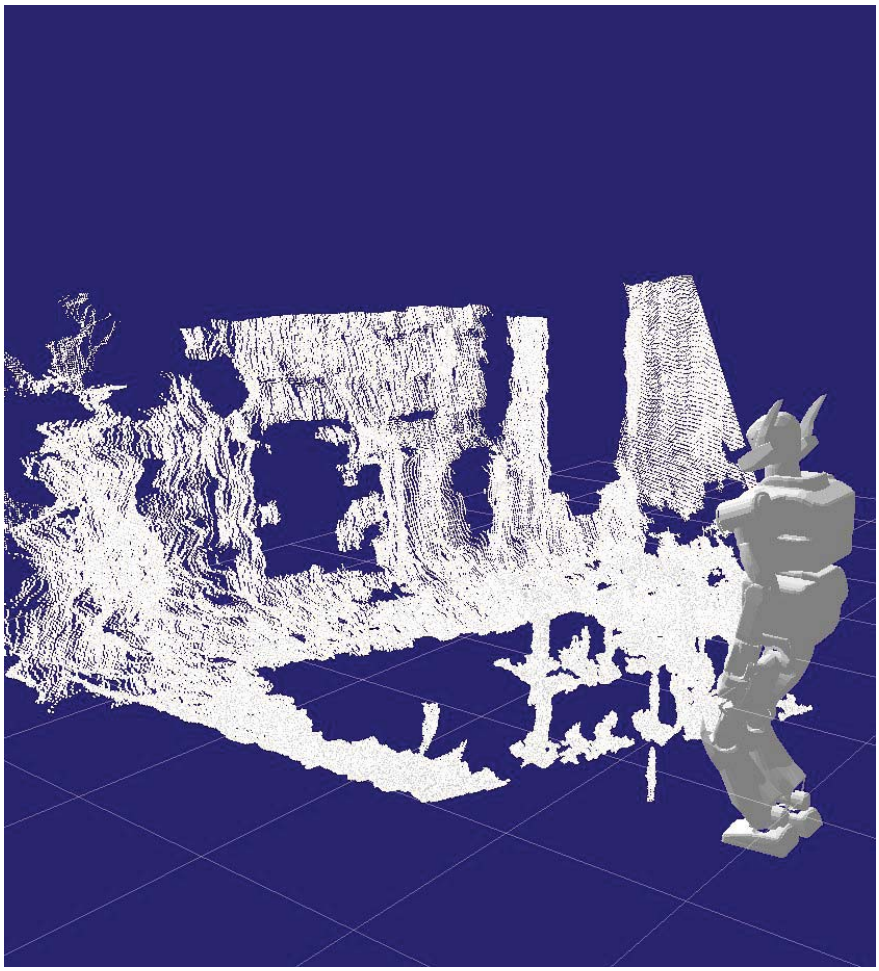

FIGURE 5.4 – Reconstruction 3D de l'environnement autour du robot HRP-2 à partir des données de la paire de caméra stéréo.

synchronisation relative de ces deux ordinateurs est, au contraire, de très bonne qualité et permet d'assurer la cohérence des données temporelles entre, d'une part, le contrôleur commandant les actionneurs et, d'autre part, les composants de décision et de perception.

Difficultés informatiques d'ordre général. Au-delà des problèmes décrits précédemment et qui affectent plus particulièrement les systèmes robotiques, les robots sont également affectés par l'ensemble des problèmes qui peuvent toucher un ordinateur : température excessive du processeur, manque d'espace de stockage, etc.

Une des difficultés de la mise en œuvre d'un système robotique est la nécessité de faire fonctionner un ensemble de technologies sachant que le mauvais fonctionnement de n'importe quel morceau de l'architecture peut mettre en péril la tâche tout entière. Qui plus est, certaines limitations du matériel tel que le couple maximum des actionneurs sont parfois vérifiées a posteriori. Des systèmes de surveillance des valeurs critiques sur les couples, les impacts sur les capteurs de force ont donc été mis en place afin d'éviter d'endommager le matériel.

5.2 Simulation transparente

Une difficulté récurrente en robotique est la lenteur de la réalisation des expérimentations : calibrer un robot et réaliser une expérience demande beaucoup de temps. Afin d'accélérer au maximum le développement et d'assurer la sécurité des expérimentateurs et du matériel, une phase de simulation des expérimentations reste cruciale.

L'utilisation d'une architecture robotique fondée sur un modèle de communication donne ici tout son intérêt : la totalité des capacités du matériel, tant au niveau de l'actionnement que des capteurs est représenté informatiquement par un ensemble composé de services et de "topics". Tant que le simulateur choisi fournit ce même ensemble de canaux de communication, les algorithmes pourront fonctionner indifféremment en simulation ou sur le véritable système. Il reste alors le plus important : assurer la cohérence entre les capteurs simulés en fonction des commandes envoyées aux actionneurs. Pour ce faire, il est important de disposer d'un moteur physique réaliste. Dans le cadre de cette thèse, le simulateur OpenHRP a été choisi. En effet, la plupart des simulateurs robotiques – Gazebo, Webots –, utilisent des moteurs physiques primitifs adaptés aux robots mobiles ne réalisant pas de mouvements "dynamiques". Afin de modéliser précisément les interactions entre le pied et le sol, un moteur physique disposant d'un modèle de contact convaincant et précis est indispensable à la simulation d'un mouvement réalisé par un robot humanoïde.

Les limitations de la plate-forme sont généralement liées à la simulation des caméras, ne pouvant jamais reproduire les difficultés inhérentes à la vision par ordinateur : changements d'illumination, délai induit par l'acquisition et le transfert des données, erreur de calibration de la caméra, etc. La simulation est donc davantage un moyen de tester la correction du raisonnement et pas tant un outil fiable pour démontrer sa robustesse à la réalité qui diverge toujours du modèle calculé.

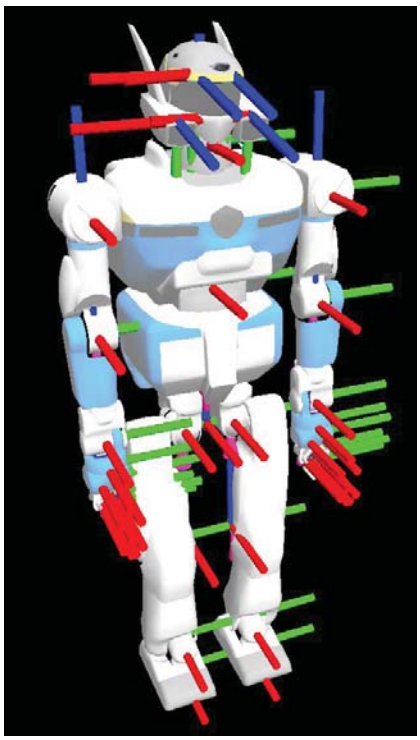

FIGURE 5.5 – Modèle de robot HRP-2 décrit via le format URDF.

5.3 Modélisation unifiée d'un système robotique

La génération de mouvement se fonde sur une connaissance précise des capacités physiques des robots. Cette description est généralement codée à l'aide d'un format de fichier permettant de représenter les différentes informations caractérisant le système. Une définition mathématique du système a été donnée dans le Chapitre 3. Du point de vue informatique, le format de fichier URDF – Unified Robot Description Format –, SRDF – Semantic Robot Description Format – et RCPDF – Robot Contact Point Definition Format – est utilisé pour représenter les informations nécessaires à la génération de mouvements sur un robot humanoïde. Les deux premiers formats sont le fruit du travail de la communauté ROS tandis que le dernier a été développé dans le cadre de cette thèse.

5.3.1 Description d'un robot

Un robot est un ensemble de corps dont les mouvements relatifs sont contraints. La modélisation des contraintes passe par la modélisation des articulations reliant deux corps entre eux. Le format URDF définit donc deux grands types d'objets : les corps et les articulations. Nous allons voir quelles informations sont attachées à chaque type d'objet.

Corps Un corps est défini par une forme géométrique. Cette forme peut soit être définie par une primitive géométrique commune – sphère, cylindre, boîte – ou bien via un modèle 3D. Les informations de la dynamique du corps sont également codées : position du centre de masse au sein de l'objet ainsi que la matrice d'inertie associée au corps. Une représentation alternative de la géométrie du corps utilisée pour les calculs des collisions peut également être définie. Dans le cas d'HRP-2, une version alternative des modèles des corps utilisant une approximation convexe avec un pas de dicrétisation plus grossier est disponible afin de pouvoir accélérer l'évaluation des collisions. Une version alternative fondée sur l'utilisation de capsules est également utilisée afin de fournir des tests de collision rapides pour la planification de mouvements.

Articulation Une articulation lie deux corps tout en imposant un ensemble de contraintes au mouvement relatif de ces deux éléments. Un joint comporte donc un lien vers le corps "père" et le corps "fils" pour définir sa position dans l'arbre cinématique. Dans le cas du joint racine, il peut ne pas y avoir de corps "père" et dans le cas d'un organe terminal, il peut ne pas y avoir de corps "fils".

Les joints supportés sont les suivants :

Libre. Ce joint virtuel possède six degrés de liberté et n'impose aucune contrainte.

Rotation. Ce joint possède un degré de liberté unique et impose un mouvement de rotation autour d'un axe spécifique au joint.

Translation. Ce joint possède un degré de liberté unique et impose un mouvement de translation le long d'un axe spécifique au joint.

Planaire. Ce joint possède deux degrés de liberté et impose un mouvement coplanaire à un plan spécifique au joint.

Fixe. Ce joint ne possède pas de degré de liberté et impose un mouvement rigide entre les deux corps.

Les degrés de liberté des joints possèdent généralement un minimum et maximum qui sont, soit le résultat de la conception mécanique du joint, soit lié à la géométrie du robot : une valeur différente entraînerait immédiatement une autocollision et il n'est donc pas intéressant de considérer ces valeurs. Chaque joint comporte donc ces valeurs ainsi qu'une vitesse et une force ou un couple maximum selon le type de joint. Enfin la friction statique du joint ainsi que l'amortissement utilisé pour la simulation de ce dernier peuvent également être enregistrés.

Capteurs Une fois l'arbre cinématique défini et annoté par les corps du robot auquel ils sont attachés, il reste encore un élément clé à spécifier : la position des capteurs. Pour ce faire, des joints fixes définissent la position de corps "virtuels" sans géométrie permettant de spécifier la position de points importants du robot tels les capteurs.

Il n'y a malheureusement pas de description générique de ces derniers qui permettrait de ne pas abuser de la structure cinématique du robot.

5.3.2 Modélisation des préhenseurs du robot HRP-2

La complexité du processus de génération de mouvements pour un système croît avec le nombre de degrés de liberté. Par exemple, HRP-2 est un système

possédant 40 degrés de liberté, dont 10 pour les mains. Cependant, tous ces degrés de liberté ne sont pas commandables. En effet, chaque main n'est commandée que par un seul degré de liberté décrivant le niveau de fermeture de la pince. Ces mécanismes sont régulièrement utilisés dans les robots. On peut également citer le robot Romeo d'Aldebaran Robotics [2] qui s'appuie sur un mécanisme de commande similaire : un degré de liberté commande l'actionnement d'une main complète composé de plusieurs degrés de liberté. Pour modéliser un tel système, il est possible d'insérer des contraintes et des degrés de liberté "dépendants" dans le modèle. On peut ainsi insérer une contrainte qui va lier deux degrés ensemble. Par exemple dans la définition du joint A, on peut spécifier que la valeur du degré de liberté associé dépend de la valeur du degré de liberté du joint B :

$$\mathbf{X}_B = \alpha_B \mathbf{X}_A + \beta_B \tag{5.1}$$

Dans l'équation précédente, α_B et β_B sont des constantes propres au joint B. $\mathbf{X}_A$ et $\mathbf{X}_B$ la valeur du degré de liberté associé respectivement aux joints A et B.

5.3.3 Prise en charge des autocollisions

Une autocollision consiste en la collision de deux corps du robot. C'est particulièrement facile sur un humanoïde : les bras et les jambes formant quatre chaînes distinctes pouvant entrer en collision les unes avec les autres. D'autres cas sont également possibles : par exemple la jambe de certains robots HRP-2 peut entrer en collision avec le bassin dans certaines configurations. De manière générale, toute collision qui requiert le mouvement de plusieurs degrés de liberté combinés et qui ne peut donc pas être évitée en spécifiant les bornes des degrés de liberté est une autocollision qui doit être prise en compte.

Pour déterminer les autocollisions, deux stratégies sont possibles. L'approche automatique consiste à échantillonner l'espace des configurations du robot afin de trouver les corps qui sont, soit toujours en collision, soit jamais en collision. Dans ces deux cas, la paire de collision est rejetée. Toutes les autres sont ajoutées. La seconde stratégie consiste à forger la liste à la main. Dans le cas où les autocollisions doivent être vérifiées dans la boucle temps réel, cette technique est souvent utilisée, car les contraintes de temps de calcul sont trop fortes pour considérer toutes les paires. On se concentre alors sur les cas les plus probables. En particulier, certains cas semblent difficiles à détecter automatiquement, notamment si pour entrer en collision avec le corps A, il faut forcément traverser le corps B alors il est inutile de prendre en compte le corps A lors de la vérification des autocollisions.

La liste des paires de collisions est enregistrée dans un fichier séparé utilisant le format SRDF – Semantics Robot Description Format –. Ce fichier stocke également la position de référence du robot dans lequel il se situe lors du démarrage des expérimentations.

2. Description officielle du prototype de plate-forme : `http://projetromeo.com/`

5.3.4 Définition des contacts autorisés

Un défi ouvert pour la robotique humanoïde dans les prochaines années est la réalisation de mouvements asservis sur des sols non plans ou utilisant des contacts avec d'autres zones du corps que les pieds. Pour un robot donné, les contacts sont autorisés à différents endroits, typiquement, les mains et les pieds. Il est donc nécessaire de définir des "zones" où les contacts sont autorisés. Cette information n'était pas formalisée jusqu'à présent dans l'architecture proposée par ROS et a été le résultat d'un travail réalisé durant cette thèse. Le format de fichier RCPDF – Robots Contact Point Description Format – associe à chaque corps 0, 1 ou plusieurs zones de contact. Chaque zone de contact peut être, soit représentée par une primitive géométrique, soit par un modèle 3D. Les normales du modèles permettent de définir la direction dans laquelle le contact peut s'effectuer. Un contact est valide si la force de réaction de l'environnement sur le corps du robot est colinéaire avec la normale de la zone de contact mais de direction opposée. Enfin, la force de réaction maximale en un point autorisée sur cette zone de contact peut être enregistrée afin d'assurer que le contact ne va pas compromettre l'intégrité structurelle du robot. Cette représentation offre une formulation générique qui peut être utilisée pour réaliser des prises de décisions moins arbitraires que celles qui sont uniquement fondées sur l'anthropomorphisme. Par exemple, la force applicable sur les mains du robot peut être plus contrainte ce qui peut pousser un solveur à faire marcher le robot "sur ses deux jambes" sans avoir à le lui spécifier de manière directe.

En pratique, ces données sont utilisées dans les algorithmes de marche bipèdes et les systèmes de surveillance afin de calculer les tailles des semelles du robot – la zone de contact autorisée sous chaque pied – et, de fait, les contraintes auxquelles sont soumises le ZMP.

5.3.5 Adaptation du modèle pour le contrôle et la planification

Instrumenter le modèle pour y incorporer le maximum d'informations est intéressant, cependant utiliser la chaîne cinématique pour positionner des capteurs pose des problèmes pratiques. En effet, l'évaluation de l'arbre est nécessaire pour de nombreux calculs qu'ils soient géométriques, cinématiques ou dynamiques, inverses ou directs. La complexité de ces algorithmes est donc dépendante de la taille de l'arbre et la présence d'articulations, même fixes, pour positionner les capteurs crée un coût supplémentaire qui semble inutile. Une proposition est en cours pour modéliser d'une manière plus propre les capteurs dans le format URDF, mais à l'heure actuelle il semble intéressant de pouvoir réaliser des simplifications ou "élagages" de l'arbre pour pouvoir en retirer les éléments qui ne sont pas nécessaires. Dans le cas où les corps que l'on souhaite supprimer sont des nœuds terminaux de l'arbre sans information dynamique – comme les capteurs –, alors l'opération peut être réalisée directement. Cependant, dans tous les autres cas, il est nécessaire de pouvoir calculer la masse et l'inertie équivalente de l'ensemble des corps rendus solidaires par la simplification opérée. De fait, ces transformations de l'arbre sont un moyen bien plus efficace d'exprimer les contraintes égalités sur certains degrés de liberté que leur insertion en tant que contrainte dans l'algorithme de génération de mouvement proprement dit. Les transformations dynamiques de l'arbre sont également nécessaires pour

adapter la géométrie du robot lorsqu'il est nécessaire, par exemple, de réaliser une tâche de locomotion en transportant un objet possédant un poids non négligeable pouvant interférer dans la dynamique du mouvement.

Le système de représentation du robot peut d'ores et déjà coder la totalité des informations proposées dans cette section, mais il n'a pas été possible, à l'heure actuelle, d'intégrer le mécanisme de transformation de l'arbre aux logiciels de planification et de contrôle utilisés sur le robot humanoïde HRP-2.

5.4 Applications robotiques

L'ensemble des outils décrits dans la section précédente doit être mis en œuvre conjointement afin de pouvoir former une architecture robotique cohérente. Chaque composant faisant partie d'une architecture robotique a beau disposer d'un modèle de communication cohérent, sembler parfaitement fonctionner pris à part et malgré tout l'assemblage du tout reste une opération délicate : a-t-on suffisamment de ressources pour faire fonctionner tous les composants à une vitesse raisonnable ? A-t-on une bande passante suffisamment importante entre les ordinateurs pour communiquer les informations transmises ? Le délai de transmission perturbe-t-il l'architecture ? La totalité de ces questions ajoutées aux difficultés inhérentes au déploiement d'une application logicielle distribuée nécessite une grande rigueur dans le développement des démonstrations. Nous allons ici développer un exemple de scénarii puis montrer quel bilan on peut tirer des séries d'expériences réalisées durant cette thèse.

5.4.1 Mise en place d'une application

Nous allons présenter dans cette section l'architecture de la démonstration robotique présentée dans la section 4.5 : le robot HRP-2 doit pouvoir réaliser une tâche de locomotion pour se déplacer jusqu'à un meuble dans lequel il souhaite déposer un objet – dans cet exemple une balle rose –. Pour ce faire, il doit tout d'abord planifier une trajectoire corps complet, c'est-à-dire combinant à la fois locomotion et manipulation. Une fois cette trajectoire déterminée, il doit l'exécuter tout en corrigeant sa trajectoire afin d'atteindre son but. La localisation est réalisée grâce aux caméras du robot.

Le système est composé des composants robotiques suivant répartis sur deux ordinateurs – un pour le contrôle et l'autre pour la décision et la perception – :

Ordinateur dédié au contrôle.

Contrôle. Ce nœud est composé d'un solveur fondé sur le paradigme de la pile de tâches et d'une infrastructure logicielle utlisant le langage de programmation Python pour insérer et définir des tâches en cours d'exécution. Les références des tâches peuvent être fournies par l'extérieur via des topics. Ce composant fonctionne partiellement en temps réel.

Export des informations capteurs et de la commande[3]**.** La commande envoyée au système, la commande après stabilisation et l'état actuel du robot lu sur les encodeurs est publié sous la forme de topics. L'accélération linéaire et la vitesse angulaire du torse du robot sont

également publiés sous la forme d'un topic. Ce composant fonctionne partiellement en temps réel.

Ordinateur dédié à la décision et à la perception.

Acquisition des images. [3] Capture les images venant de la paire stéréo grand angle.

Conversion de l'espace couleur. Transforme les images encodées en Bayer vers des images monochromatiques.

Rectification. Corrige la déformation de l'image liée à l'utilisation de l'objectif grand angle et à la conception de la caméra.

Localisation. Réalise une estimation de la position du robot dans le monde en utilisant les caméras du robot.

Cinématique directe. À partir de l'état des encodeurs, ce composant recalcule la position relative de tous les corps et les publie sous forme de topics.

Évaluation de l'erreur de positionnement. Compare la position voulue du robot dans le plan à un instant t à la position perçue au même instant.

Divers. Plusieurs nœuds supplémentaires sont chargés de la surveillance des paramètres critiques du robot.

Ces composants interagissent entre eux et fournissent chacun des flux de données et des services qui sont détaillés dans la Figure 5.6.

Cette architecture a été testée sur le robot humanoïde HRP-2. L'architecture a fonctionné correctement, mais il n'a pas été possible d'atteindre une précision suffisante au sein du système de perception pour compenser avec succès la dérive du robot. En effet, le SLAM tel que configuré ici a donné une précision de plus ou moins 5cm ce qui est insuffisant pour des tâches de locomotions complexes. On notera également que la phase de planification n'apparaît pas, elle a été réalisée grâce au travail décrit dans Dalibard et collab. (2011) sous la forme d'un traitement hors ligne. Le chemin est enregistré sous la forme d'un fichier chargé au début de l'expérimentation.

5.4.2 Difficultés récurrentes

La mise en place d'une architecture modulaire pour la robotique sert un objectif principal : contenir la complexité en assurant à l'utilisateur la possibilité de pouvoir s'abstraire d'une partie des traitements sous-jacents, quitte à obtenir une solution sous-optimale. En effet, l'agrégation d'outils aussi différents que la planification de mouvement, le contrôle, l'asservissement et différentes techniques de perception nécessite de pouvoir s'appuyer sur des composants aussi automatiques que possible. En particulier, concevoir des algorithmes ne nécessitant pas de paramétrage poussé est important, mais également très difficile. Le paramétrage peut passer, soit par une phase de calibration – comme souvent en perception –, une phase de construction de carte, par exemple pour la navigation – ou le réglage des stratégies adoptées par les algorithmes. Les outils de planification nécessitent souvent le réglage de paramètres ce qui complexifie leur utilisation. Si l'on ne peut paramétrer le système automatiquement, il faut

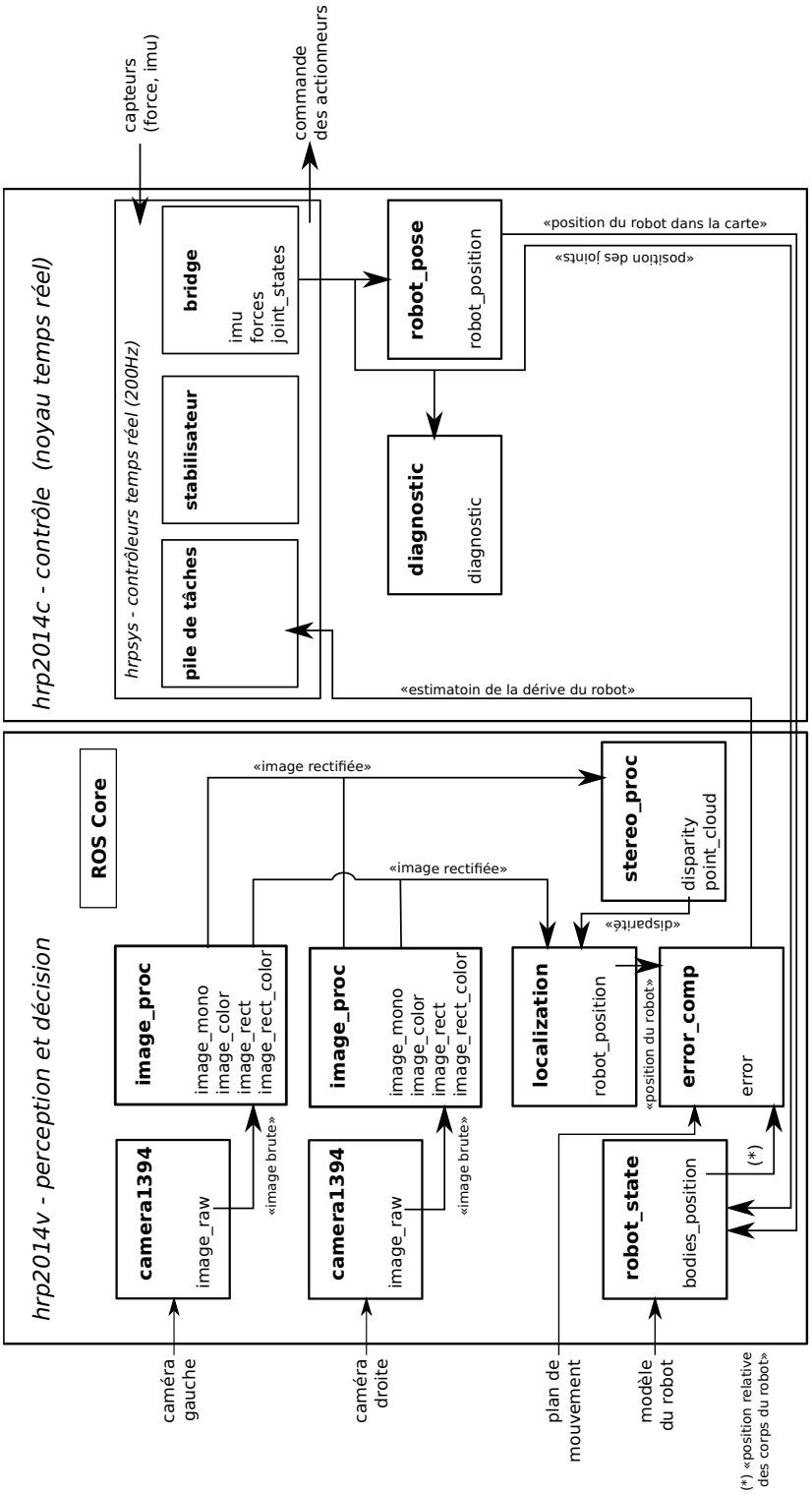

FIGURE 5.6 – Schéma de fonctionnement de l'architecture.

alors tenter de trouver un ensemble de paramètres ayant une réalité physique qui puisse permettre à l'utilisateur de se construire plus facilement un modèle de comportement de l'algorithme.

Une seconde difficulté concerne l'interprétation des données. La majorité des algorithmes produisent en sortie des données numériques possédant un sens physique. Prenons par exemple un composant réalisant le suivi d'un objet en temps réel : il va alors fournir en sortie la position de l'objet dans l'espace euclidien 3D. Une ambiguïté se pose alors : par rapport à quel repère référence cette donnée est-elle exprimée ? La caméra du robot ? La racine de sa chaîne cinématique ? Un point fixe du monde ? Face aux nombreux dysfonctionnements provoqués par ce type de problème, ROS fournit une solution clé en main : chaque transformation précise, sous forme de chaîne de caractère, de quel repère à quel repère cette donnée est la transformation. En enregistrant ces données, on acquiert également la possibilité de calculer automatiquement des transformations d'un repère à un autre. Par exemple si l'on fournit la transformation de A à B et de B à C, on peut demander la transformation de A à C et elle peut être calculée automatiquement. Il y a là un véritable gain puisque l'on passe d'une implémentation dépendant directement des données des composants auquel on est lié vers une requête purement sémantique : "quelle est la position relative de la cheville gauche par rapport au poignet droit ?". En réalisant une résolution automatique des transformations, on se préserve également dans le cas où les composants utilisés changeraient leur repère de référence entre deux versions. Une fois de plus, l'absence de typage ici rend le diagnostic difficile : aucune erreur de compilation n'est possible et les erreurs dans les valeurs numériques elles-mêmes sont toujours difficiles à détecter. On notera que le système adopté ici se fonde sur le traitement des chaînes de caractères ce qui est plus lent que le recours au typage direct tel que proposé dans Laet et collab. (2012).

Une troisième difficulté est la robustesse aux phases transitoires. Tout système complexe possède une phase d'initialisation dans laquelle le comportement des modules peut être sous-optimal. En particulier quand plusieurs composants communiquant ensemble sont lancés en même temps, il n'y a aucune garantie que le premier composant terminant son initialisation soit toujours le même. De ce fait, un composant peut commencer à attendre des données sur un topic avant que le topic ne soit créé par le premier composant. Le protocole de communication est extrêmement souple sur ce point et la possibilité de pouvoir écouter des flux de données qui n'existent pas encore simplifie de beaucoup le développement en évitant les erreurs lorsque l'ordre d'initialisation change. Par contre, il est alors nécessaire de mettre en place, en interne, les mécanismes afin d'éviter les situations de famine où plusieurs composants s'attendent les uns les autres indéfiniment. Cependant, le graphe de communication étant le plus souvent un arbre, ce genre de cas reste peu courant.

La quatrième difficulté concerne l'initialisation du système. Il est souvent difficile de lancer une démonstration en "un clic", mais plus le nombre d'étapes pour démarrer la démonstration est important, plus le risque d'introduire une erreur humaine lors d'une expérimentation augmente. Il est important de limiter les opérations pour démarrer une démonstration. Un outil utilisé dans le cadre de ce travail est un formalisme permettant de décrire informatiquement – en XML –, le graphe formé par l'ensemble des composants robotiques ainsi que leurs paramètres respectifs. Ce système réduit donc au maximum les erreurs humaines, mais certaines difficultés persistent : un robot humanoïde utilisant

une commande à gains forts telle que HRP-2 s'initialise en l'air, puis il doit être posé, le système possède donc plusieurs phases transitoires d'initialisation qu'il est difficile d'automatiser.

FIGURE 5.7 – Visualisation de l'état général du robot fourni par les outils de diagnostic automatique.

Le dernier problème récurrent est la difficulté pour un utilisateur de déterminer si le système est dans un état stable et, si ce n'est pas le cas, arriver à diagnostiquer le système. La section précédente a développé un ensemble d'outils permettant de surveiller l'état du robot tant au niveau mécanique, informatique que de l'utilisation qui en est faite en temps réel – vérification des couples au niveau des actionneurs notamment –. Cependant, les problèmes ne viennent pas que de ces composants "bas-niveau". Des composants algorithmiques peuvent également échouer pour diverses raisons. Pour aider la compréhension de l'état du système, le premier élément est d'unifier les mécanismes de rapport d'état. En effet, la plupart des bibliothèques complexes viennent avec leur système de journalisation qui n'est pas toujours adapté. Dans le cadre de cette architecture, nous avons la possibilité de pouvoir agréger les informations dans un seul topic qui peut ensuite être observé pour connaître l'évolution du système. On a alors la liste des messages envoyés par les différents composants dans l'ordre chronologique ce qui n'est pas le cas quand chacun fonctionne de manière séparée. Les causes et les conséquences sont, dans ce contexte, bien plus faciles à identifier. Le deuxième niveau consiste à publier sur le topic de diagnostic l'état du nœud. Cela correspond à l'un des états suivants : OK, erreur, avertissement auxquels on associe des données techniques pour aider au diagnostic, voir Figure 5.7. Enfin, un service standardisé peut également être fourni par un composant pour qu'il tente un autodiagnostic. Ces mécanismes permettent d'aider à la compréhension des erreurs.

5.4.3 Bonnes pratiques pour la robotique

Cette section détaille la méthodologie à suivre tant pour développer un composant robotique que pour développer une architecture robotique complète.

Pour développer un composant, un exemple de développement réalisé pendant cette thèse a été choisi : il s'agit de l'intégration d'un logiciel de suivi. Le cycle de développement d'une architecture robotique est ensuite détaillé.

Étude de cas : intégration d'un algorithme de suivi

Le composant dont il est question ici est librement disponible sur internet. Il s'agit de la stack ROS `vision_visp`[4] et plus particulièrement du paquet `visp_tracker`[5].

L'algorithme de suivi d'objet a été développé au sein de l'équipe LAGADIC[6] – INRIA Rennes-Bretagne Atlantique et à l'IRISA–. Cet algorithme utilise l'asservissement visuel pour calculer une estimation de la pose d'un objet. Le modèle de l'objet est connu et une estimation initiale de la position de ce dernier est nécessaire. Une fois cette estimation connue, l'algorithme tente de minimiser l'écart entre la position des lignes – bords – de l'objet dans l'image par rapport à la reprojection du modèle de l'objet suivi dans l'image. D'autres critères que les lignes peuvent être choisis, mais c'est ce dernier point qui a été retenu dans nos tests. La résolution de ce problème passe par la méthode de Levenberg-Marquardt qui est proche de l'algorithme des moindres carrés classiques.

Cet algorithme est partie intégrante de ViSP – Visual Servoing Platform –[7]. ViSP intègre directement des outils pour acquérir des images à partir de caméras ou lire des données préenregistrées, mais n'est pas un composant robotique en lui-même.

La première étape a été de concevoir l'interface du composant : flux de données et services. L'information calculée par ce programme est simple à définir : il s'agit de la position relative de l'objet suivi par rapport à la caméra. On peut donc ajouter un repère supplémentaire à l'annuaire de transformations afin de spécifier la position de l'objet suivi. Il sera dès lors possible de calculer, par exemple, la position relative du poignet par rapport à cet objet pour tenter de l'atteindre, s'il existe une chaîne entre de transformations entre ces deux repères. En entrée, le problème est plus complexe. Il faut d'une part un flux d'images, des informations de calibration, un modèle 3D de l'objet à suivre et un jeu de paramètres permettant de régler l'algorithme de suivi ainsi qu'une pose initiale.

Il est courant de modéliser le comportement des composants robotiques sous la forme d'une machine à état. Dans notre cas le composant de suivi est dans un état "non initialisé" au départ où il reçoit déjà des images et des informations de calibration, mais ne réalise aucun traitement. Un service permet d'initialiser le suivi en fournissant à la fois la pose initiale de l'objet, son modèle et les paramètres de suivi. Le nœud passse alors dans un état actif où il réalise le suivi. Si le suivi est perdu à un moment ou un autre, le nœud passe dans un état "perdu" où il tente de réinitialiser le suivi avec la dernière position connue. Si une estimation de la position de la caméra est disponible, elle va être utilisée pour tenter de fournir une estimation de la nouvelle position de l'objet en considérant ce dernier statique. À n'importe quel moment le service d'initialisation permet de relancer le suivi.

4. Site officiel : `http://www.ros.org/wiki/vision_visp`
5. Site officiel : `http://www.ros.org/wiki/visp_tracker`
6. Site de l'équipe LAGADIC : `http://www.irisa.fr/lagadic/`
7. Site du logiciel ViSP : `http://www.irisa.fr/lagadic/visp/visp.html`

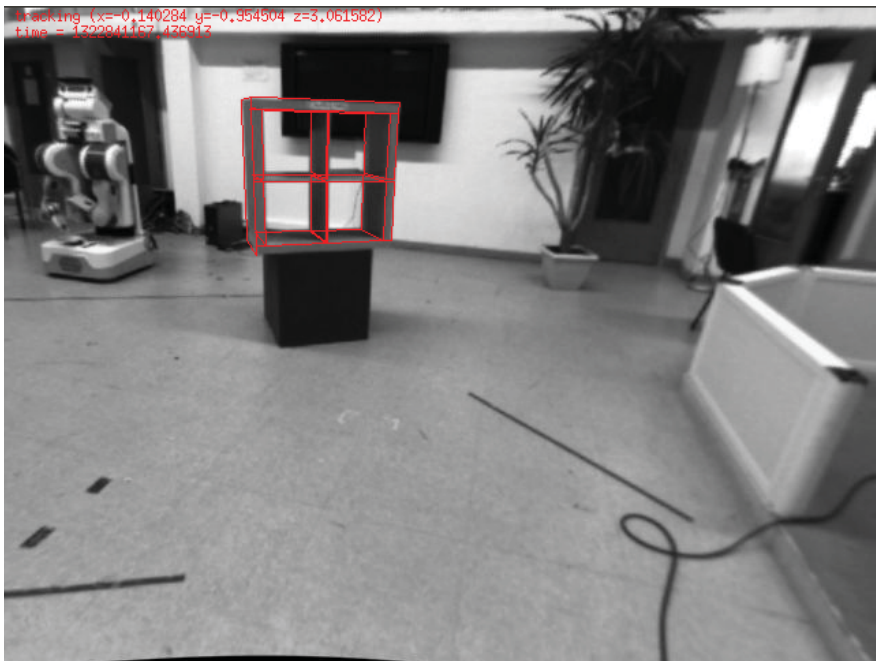

FIGURE 5.8 – Le visualisateur associé au composant de suivi d'objet.

Afin de simplifier l'utilisation du nœud, deux programmes additionnels sont fournis : le premier permet de calculer une pose initiale de manière graphique. L'utilisateur peut cliquer sur des points de l'objet prédéfini et dont on connaît la position relative en trois dimensions dans l'objet. Cet outil transmet également au composant de suivi le modèle de l'objet. Le contexte distribué de l'application empêche de passer directement des emplacements de fichiers, car il n'y a aucune garantie que le composant réalisant le suivi et le programme réalisant l'initialisation partage leur système de fichier. Le modèle est donc directement transmis sous sa représentation textuelle via le réseau. De plus, le composant pouvant être lancé à distance, laisser l'utilisateur spécifier le chemin vers le modèle sous la forme d'un chemin local est une mauvaise idée, car il sera tenté d'y insérer le chemin sur sa machine et non sur celle où le composant va être lancé. Pour ce faire, l'emplacement du modèle est passé sous la forme d'une URI – Uniform Resource Identifier – (Berners-Lee et collab., 1998).

Le second outil permet de surveiller l'état du tracking en affichant non seulement la reprojection du modèle dans l'image, mais surtout l'état du suivi des lignes du modèle. L'algorithme pouvant rejeter des points pour différentes raisons, cet outil fournit un retour graphique de ces informations afin d'aider l'utilisateur à trouver le meilleur ensemble de paramètres pour optimiser le suivi.

Le dernier point à considérer a été d'intégrer le modèle de calibration de ROS à ViSP. En effet, ROS annote chaque image avec les paramètres intrinsèques de caméra. Ces données sont exprimées sous forme matricielle et doivent être converties avant d'être mises à jour dans ViSP ce qui nécessite un traitement simple. L'avantage étant que de cette manière, on peut complètement contourner

le processus de rectification de ViSP qui nécessiterait une calibration spécifique pour l'utilisation de ce composant.

Les difficultés rencontrées durant cette intégration sont nombreuses : il a fallu supprimer le code interactif, notamment pour calculer la pose initiale qui nécessite de cliquer sur une image. Un tel code ne peut pas fonctionner sur le robot. Il a fallu s'abstraire du système de fichier local afin de rendre le système facilement distribuable et il a fallu trouver une façon de pouvoir exprimer les informations de "debug" de manière minimale. Une solution aurait été de tracer une image avec le résultat du suivi, mais les robots étant souvent reliés en WiFi à la console le contrôlant, transmettre un tel flot d'information n'est pas justifiable. À la place, un topic supplémentaire pour les développeur a été ajouté. Le défi est alors d'assurer la cohérence des données entre d'un côté les images provenant du composant de vision, de la pose provenant du tracker ainsi que du topic transportant les informations supplémentaires pour les développeurs. Ces informations sont filtrées afin que seul les triplets présentant un temps identique soient affichés. En effet, dans les précédentes versions, cette synchronisation n'était pas réalisée. On avait alors l'impression que le suivi était de mauvaise qualité alors que le problème venait simplement de l'affichage. Les paramètres de tracking sont également modifiables durant l'exécution afin de simplifier la recherche du meilleur paramétrage. Cela ouvre également la possibilité pour un composant de plus haut niveau, de changer le comportement du logiciel de suivi et réaliser de ce fait, une forme de suivi "supervisé". On notera qu'une fois de plus, la définition des paramètres joue un rôle très important. Ici l'absence de sens physique de certains d'entre eux est dommageable : par exemple la recherche de la nouvelle pose se fait dans un voisinage qui est défini en pixel. En pratique, cela signifie que si l'on se rapproche de l'objet, l'absence de reconfiguration de ce paramètre va causer l'échec du suivi. Un meilleur paramètre aurait été de passer par une borne sur la vitesse maximum de la caméra qui aurait permis d'avoir un comportement cohérent quelque soit la distance à l'objet.

Au final, ce composant a reçu un accueil favorable de la communauté avec plusieurs utilisateurs confirmés dans des laboratoires et sociétés tiers. Ces utilisateurs n'avaient pas d'expérience avec les techniques d'asservissement visuel avant de tenter d'intégrer ce composant et ont réussi à obtenir des résultats satisfaisants dans le cadre de leur utilisation sur des robots différents, notamment le robot REEM de Pal Robotics [8]. De ce fait l'objectif qui consistait à pouvoir maîtriser la complexité de la mise en place d'un tel mécanisme a été rempli au travers des choix réalisés ici. Ce composant a pu être utilisé dans le cadre d'une architecture robotique distribuée sur deux ordinateurs et l'introduction d'une interface au niveau de la componentisation robotique a permis au robot HRP-2 de se localiser, soit en utilisant ce composant, soit en utilisant un composant de localisation utilisant des amers visuels, soit encore le système de capture de mouvement de manière transparente. Seul le lancement du graphe diffère dans ces trois cas.

Cycle de développement d'une architecture

La section précédente a décrit le développement d'un composant en particulier, mais concevoir une architecture ne revient pas à simplement dupliquer cette

8. Site officiel : `http://www.pal-robotics.com/`

approche autant de fois qu'il y a de composants nécessaires. Il faut tout d'abord découper le problème en composants cohérents, puis assigner ces composants à un ordinateur parmi ceux qui équipent le robot et enfin paramétrer chacun des composants de manière optimale. Dans la mesure où une approche modulaire a pour intérêt principal la réutilisabilité des composants, on sera donc contraint par l'interface des composants préexistants que l'on devra réutiliser, ainsi que par les contraintes matérielles de la plate-forme : les caméras d'HRP-2 sont connectées à un PC donné donc le composant d'acquisition des images ne peut fonctionner que sur ce dernier. Inversement, les commandes sont envoyées via une carte connectée au second PC, le composant de contrôle ne pourra tourner que sur celui-ci. De plus, le temps réel contraint le contrôleur a n'être constitué que d'un seul processus. Le dernier élément à prendre en compte est la communication intra-nœuds : les communications par "topics" ou service induisent un retard qui doit pouvoir être accepté ou géré convenablement. Un autre élément à prendre en compte est la bande-passante disponible entre les ordinateurs qui constituent le robot. Il faut s'efforcer de minimiser la bande passante consommée entre les ordinateurs. Pour les composants effectuant de l'affichage ou de la surveillance distant le problème est souvent encore plus critique dans la mesure où le robot est la plupart du temps connecté en wi-fi et dispose donc d'un lien avec une bande-passante extrêmement limitée vers l'extérieur. Cela va donc contraindre les composants de perception, la plupart du temps gros producteurs et consommateurs de donnée, à se situer sur le même hôte. Une fois le traitement des images terminé, le résultat final peut représenter un volume d'informations bien plus compact : trajectoire ou position 3D par exemple.

En considérant la répartition des composants sur les différentes machines, il faut déterminer la liste des composants nécessaires. En général, on part d'un côté de la liste des capteurs que l'on souhaite utiliser, que l'on fait communiquer avec les algorithmes utilisant ces données en entrée qui eux-mêmes communiquent avec la partie décisionnelle. Cette partie étant généralement algorithmique, le positionnement de ces composants n'est pas soumis à des contraintes techniques fortes. Enfin, un composant de contrôle est nécessaire pour envoyer les commandes.

Une fois les nœuds choisis et paramétrés, l'étape suivante est la simulation. De nombreux outils permettent de simuler un mouvement dynamique dans un monde virtuel. Il est nécessaire ici de remplacer les composants réalisant l'acquisition des données capteurs par des composants accédants aux informations équivalentes simulées et le contrôleur par un contrôleur envoyant sa commande au simulateur. Une phase supplémentaire de définition du monde est souvent nécessaire.

Une fois validée en simulation, l'architecture est portée sur le robot proprement dit. On peut alors faire tourner l'architecture sans envoyer réellement les commandes aux actionneurs afin de vérifier l'accès aux données capteurs, les délais induits par les communications entre les hôtes, etc. Enfin, l'architecture peut être testée réellement en validant plusieurs scénarii adaptés de difficulté croissante. Une fois l'expérience validée, on peut alors changer certains composants afin de tester de nouvelles stratégies ce qui fait débuter un nouveau cycle.

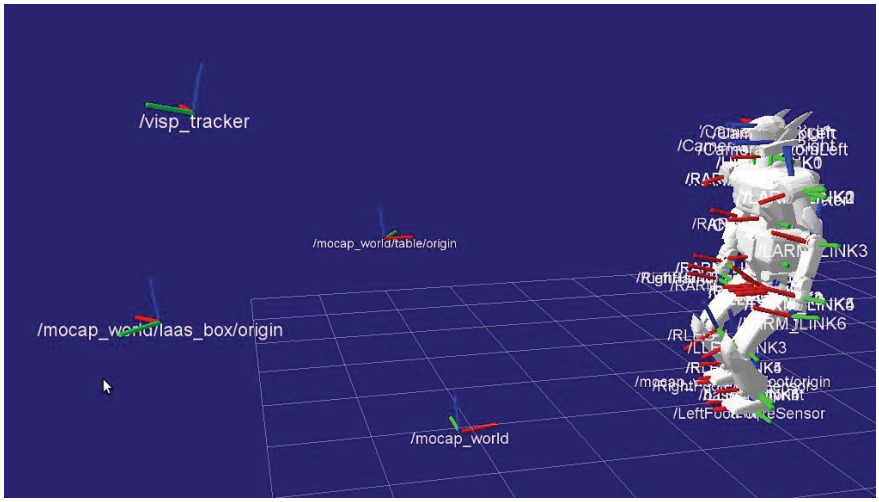

FIGURE 5.9 – Architecture associant de nombreux composants robotiques intégrant le contrôleur du robot exécutant une tâche de locomotion, le système de capture de mouvement et le processus complet de vision utilisant l'algorithme de suivi décrit dans cette section pour estimer la position d'un cube.

5.5 Conclusion

Construire une architecture robotique est une tâche ardue, non seulement parce qu'elle agrège des algorithmes complexes, mais surtout parce qu'il est nécessaire de considérer la plate-forme utilisée et les problèmes techniques qui lui sont propres – délais, puissance de calcul disponible –, afin de construire un ensemble fonctionnel. Cette section a donc à la fois présenté comment développer un composant robotique puis comment combiner plusieurs composants pour réaliser une architecture. Les différentes difficultés inhérentes à la conception d'un système ont été passées en revue ainsi que les stratégies qui ont été adoptées au cours de ces trois années. Il est probable qu'au cours des prochaines années de nombreuses architectures robotiques voient le jour sur des robots humanoïdes afin de les rendre sensibles à l'évolution de leur environnement. Un objectif ambitieux pour lequel les robots mobiles restent la plate-forme d'expérimentation principale dans l'état actuel de la littérature.

6 Conclusion

Can we actually "know" the universe? My God, it's hard enough finding your way around in Chinatown.

Woody Allen
My Philosophy

6.1 Conclusion

6.1.1 Bilan...

Les travaux présentés dans ce manuscrit de thèse ont pour but d'améliorer les algorithmes de génération et d'exécution de mouvements pour les robots humanoïdes. Le premier travail présenté a traité de l'amélioration de la représentation des problèmes d'optimisation afin de faciliter le prototypage et l'écriture de solveurs, tout en assurant un haut niveau de sécurité. Un solveur ne doit accepter que des problèmes qui lui sont adaptés. Une boîte à outils de fonctions de coût et de contraintes pour l'optimisation de trajectoires a également été conçue. L'effort théorique principal a été de chercher une interface telle que la théorie de l'optimisation numérique se trouve exprimée dans sa totalité tout en excluant les problèmes mal construits et en maitrisant le coût algorithmique de la résolution des problèmes. Par exemple, certains gradients ou hessiens peuvent ne pas être utilisés, mais aucune approximation par différence finie ne peut être réalisée automatiquement. C'est à l'utilisateur d'explicitement instancier ce comportement, et d'assumer le coût de calcul supplémentaire généré par ce type d'approche. Un tel outil est adapté au post-traitement de trajectoires générées par des algorithmes aléatoires notamment.

Le second chapitre détaille comment générer une trajectoire de marche équilibrée sur un sol plan. Les trajectoires sont modélisées sous forme de tâches et ces dernières sont insérées dans un solveur temps-réel générant la commande. Le tout forme un contrôleur polyvalent permettant, entre autres, l'exécution de trajectoires de marche. Dans des travaux précédents, cette stratégie a déjà été appliquée pour permettre à un robot humanoïde de marcher mais jusqu'à présent ces tâches fonctionnaient en boucle ouverte et étaient donc soumises à une dérive empêchant la réalisation de trajectoires longues ou nécessitant une grande précision. L'altération du schéma de contrôle par l'utilisation d'un composant de localisation estimant cette dérive a permis la réalisation de mouvements d'une grande précision. La localisation est ici prise en charge par un système de capture de mouvements.

Cela nous a naturellement amenés à nous poser deux questions abordées dans le troisième chapitre : peut-on obtenir le même résultat en utilisant les capteurs embarqués ? Peut-on représenter de manière générique, au-delà de la simple locomotion, des trajectoires asservies ? À la première question, la réponse est, oui, il est possible de se localiser grâce aux caméras embarquées, mais au prix d'une précision moindre. Si cette qualité moindre ne gêne pas la navigation en soi, elle empêche malheureusement toute manipulation complexe une fois la trajectoire terminée. Il est donc nécessaire de non seulement asservir la locomotion, mais également la manipulation. La locomotion nécessite un système de localisation sans dérive, proche des besoins d'un robot mobile alors que la manipulation nécessite avant tout un système de localisation dont l'erreur tend vers zéro lors de la réalisation d'un contact. Un asservissement visuel par exemple est tout à fait adapté à cet objectif, car plus le robot s'approche de l'objet, plus la précision croît. Malheureusement, il n'a pas été possible de tenter une expérimentation intégrant à la fois vision embarquée, asservissement visuel sur un objet et correction combinée des trajectoires de locomotion et de manipulation.

Le dernier chapitre, quant à lui, détaille le fonctionnement de l'architecture

robotique mise en place durant cette thèse et qui a été largement partagée avec d'autres doctorants du groupe. Il est nécessaire de comprendre le pas franchis puisque l'on est passé d'une architecture réalisant, soit une lecture des trajectoires articulaires en boucle ouverte, soit la réalisation d'une pile de tâches avec un asservissement *ad hoc* dédié au scénario et difficile à généraliser vers un système fondé sur un ensemble de composants robotiques réutilisables. S'il n'y a pas eu écriture des composants robotique, un travail de conception de l'architecture a été nécessaire afin de permettre l'utilisation conjointe de toutes les capacités du robot.

6.1.2 . . . et perspectives

Dans le cadre du travail réalisé pour la modélisation générique des problèmes d'optimisation, il reste encore de nombreux travaux à mener dans trois directions différentes. La première est la représentation informatique de fonctions mathématiques afin d'étendre l'expressivité des fonctions. L'écriture de fonctions dépendant de plus d'une variable se trouve compliquée par la nécessité de pouvoir calculer les gradients combinés issus de la dérivation de plusieurs variables différentes. La représentation d'espaces mathématiques spécifiques tel que le groupe spécial Euclidien $\mathrm{SE}(n)$ imposant des contraintes tacites au solveur. La seconde direction est l'écriture de nouvelles fonctions afin de pouvoir fournir une boîte à outils plus complète : optimisation de posture, optimisation de trajectoires dynamiques, ajout de critères de stabilité tel que le ZMP, etc. La dernière direction de recherche est l'intégration de nouveaux solveurs et de nouvelle formulations de problèmes. Par exemple, l'utilisation du contrôle optimal est très courante en robotique. Ce domaine traite des fonctions de coût dont la forme est très contrainte et est donc un intéressant candidat pour être modélisé par le paradigme développé ici.

Concernant le travail d'exécution de trajectoire sur robots humanoïdes, le chemin qui reste à parcourir est clair : pour pouvoir asservir une trajectoire automatiquement, il faut encore valider la correction des trajectoires de manipulation, ainsi que l'utilisation d'algorithmes de vision pour estimer la déformation nécessaire de la trajectoire des bras. Une fois les tâches de locomotion et de manipulation asservies, il faudra alors prendre en compte ces contraintes dans la phase de planification. À l'heure actuelle, le planificateur peut générer un plan de mouvement, mais il ne prend pas en compte la visibilité des amers dans la phase de génération de la trajectoire. Il faut privilégier les chemins assurant une qualité maximale de l'asservissement. Les contraintes imposées par les systèmes de localisation variant selon les cas, un dialogue entre l'étage de planification et l'étage de perception et localisation est nécessaire. La dernière étape serait alors de ne plus se contraindre à des transitions temporelles dans le plan, mais également insérer des événements logiques. Pour ce faire, il sera nécessaire d'insérer dans l'architecture un superviseur de haut niveau permettant de réaliser la transition entre les états logiques ainsi que de gérer les retours sur erreur.

En conclusion, les travaux entrepris durant ces trois ans ont permis de poser les fondations de l'exécution de trajectoires asservies sur le robot humanoïde HRP-2, mais il reste encore un important effort à fournir pour arriver à exécuter des scénarii longs et précis de manière fiable. Il est fort probable que les années à venir voient d'importants travaux dans ce sens, le prochain défi proposé par la DARPA (Tactical Technology Office, 2012) étant destiné aux robots

FIGURE 6.1 – Le défi DARPA pour les robots humanoïdes : les robots devront, entre autres, remplacer une vanne et détruire un mur pour pouvoir y accéder.

humanoïdes – voire Figure 6.1 –. Ces derniers devront réaliser des mouvements complexes tels que monter une échelle ou bien encore conduire une petite voiture. Les techniques de fermeture de boucle comme celle qui a été proposée dans ce manuscrit joueront sans doute un grand rôle dans la réussite de scénarii de ce type.

Table des figures

Liste des tableaux

Bibliographie

Alcantarilla, P., K. Ni, L. Bergasa et F. Dellaert. 2011, «Visibility Learning in Large-Scale Urban Environment», dans *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

Alcantarilla, P., S. Oh, G. Mariottini, L. Bergasa et F. Dellaert. 2010, «Learning Visibility of Landmarks for Vision-Based Localization», dans *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, AK, USA.

Ansar, A. et K. Danilidis. 2003, «Linear Pose Estimation from Points or Lines», *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 25, n° 4, p. 1–12.

Arechavaleta, G., J.-P. Laumond, H. Hicheur et A. Berthoz. 2006, «Optimizing principles underlying the shape of trajectories in goal oriented locomotion for humans», dans *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, p. 131–136, doi :10.1109/ICHR.2006.321374.

Atkeson, C., A. Moore et S. Schaal. 1997, «Locally weighted learning», *AI Review*, vol. 11, p. 11–73.

Baudouin, L., P. Perrin, T. Moulard, O. Stasse et Y. E. 2011, «Real-time replanning using 3d environment for humanoid robot», dans *11th IEEE/RAS International Conference on Humanoid Robots (Humanoids), to appear*. URL `http://hal.archives-ouvertes.fr/hal-00601300/en/`.

Bay, H., A. Ess, T. Tuytelaars et L. V. Gool. 2008, «SURF : Speeded Up Robust Features», *Computer Vision and Image Understanding*, vol. 110, n° 3, p. 346–359.

Ben-Kiki, O., C. Evans et I. döt NetT. 2009, «YAML^{TM}Specification Index», URL `http://yaml.org/spec/`.

Berners-Lee, T., R. Fielding et L. Masinter. 1998, «Uniform Resource Identifiers (URI) : Generic Syntax», RFC 2396 (Draft Standard). URL `http://www.ietf.org/rfc/rfc2396.txt`, obsoleted by RFC 3986, updated by RFC 2732.

Bolles, R. et M. Fischler. 1981, «A RANSAC-based approach to model fitting and its application to finding cylinders in range data», dans *Intl. Joint Conf. on AI (IJCAI)*, Vancouver, Canada, p. 637–643.

Castagna, G., G. Ghelli et G. Longo. 1995, «A calculus for overloaded functions with subtyping», .

Chaumette, F. et S. Hutchinson. 2006, «Visual servo control. i. basic approaches», *Robotics & Automation Magazine, IEEE*, vol. 13, n° 4, p. 82–90.

Chaumette, F., S. Hutchinson et collab.. 2007, «Visual servo control, part ii : Advanced approaches», *IEEE Robotics and Automation Magazine*, vol. 14, n° 1, p. 109–118.

Chestnutt, J. 2010, *Motion planning for humanoid robots*, chap. Navigation and Gait Planning, K. Harada and E. Yoshida and K. Yokoi éd., Springer.

Chestnutt, J., J. Kuffner, K. Nishiwaki et S. Kagami. 2003, «Planning biped navigation strategies in complex environments», dans *3rd IEEE/RAS International Conference on Humanoid Robots (Humanoids)*.

Chestnutt, J., Y. Takaoka, K. Suga, K. Nishiwaki, J. Kuffner et S. Kagami. 2009, «Biped navigation in rough environments using on-board sensing», dans *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, p. 3543–3548, doi :10.1109/IROS.2009.5354575.

Choset, H., K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki et S. Thrun. 2005, *Principles of Robot Motion : Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents)*, The MIT Press, ISBN 0262033275. URL `http://www.worldcat.org/isbn/0262033275`.

Craig, L., J. L. Zhou et A. L. Tits. 1997, «User's guide for cfsqp version 2.5 : A c code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints», cahier de recherche, Electrical Engineering Department and Institute for System Research.

Dalibard, S., A. E. Khoury, F. Lamiraux, M. Taix et J.-P. Laumond. 2011, «Small-space controllability of a walking humanoid robot», dans *IEEE-RAS Intl. Conference on Humanoid Robots*, Bled, Slovenia, p. 739–744.

Dang, D., F. Lamiraux et J. Laumond. 2011, «A framework for manipulation and locomotion with realtime footstep replanning», dans *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, IEEE, p. 676–681.

Davison, A. J., I. D. Reid, N. D. Molton et O. Stasse. 2007, «MonoSLAM : Real-Time Single Camera SLAM», *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, n° 6, p. 1052–1067.

Dimitrov, D., A. Paolillo et P.-B. Wieber. 2011, «Walking motion generation with online foot position adaptation based on l1- and l∞-norm penalty formulations», dans *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China. URL `http://hal.inria.fr/inria-00567671/en`.

Dimitrov, D., P.-B. Wieber, O. Stasse, H. Ferreau et H. Diedam. 2009, «An optimized linear model predictive control solver for online walking motion generation», dans *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, ISSN 1050-4729, p. 1171 –1176, doi :10.1109/ROBOT.2009.5152380.

Driesen, K. et U. Hölzle. 1996, «The direct cost of virtual function calls in C++», *ACM Sigplan Notices*, vol. 31, nº 10, p. 306–323, ISSN 0362-1340.

Dune, C., A. Herdt, O. Stasse, P.-B. Wieber, K. Yokoi et E. Yoshida. 2010, «Cancelling the sway motion of dynamic walking in visual servoing», dans *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, ISSN 2153-0858, p. 3175 –3180, doi :10.1109/IROS.2010.5649126.

Erdyn Consultants. 2012, «Le développement industriel futur de la robotique personnelle et de service en france», cahier de recherche, Pôle Interministériel de Prospective et d'anticipation des mutations économiques (PIPAME).

Ferre, E. et J.-P. Laumond. 2004, «An iterative diffusion algorithm for part disassembly», dans *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 3, ISSN 1050-4729, p. 3149–3154 Vol.3, doi :10.1109/ROBOT.2004.1307547.

Foissotte, T., O. Stasse, A. Escande, P. Wieber et A. Kheddar. 2009, «A two-steps next-best-view algorithm for autonomous 3d object modeling by a humanoid robot», dans *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, IEEE, p. 1159–1164.

Gamma, E., R. Helm, R. Johnson et J. Vlissides. 1994, *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Harada, K., S. Kajita, K. Kaneko et H. Hirukawa. 2004, «An analytical method on real-time gait planning for a humanoid robot», dans *4th IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, vol. 2, p. 640–655, doi :10.1109/ICHR.2004.1442676.

Harris, C. et M. Stephens. 1988, «A combined corner and edge detector», dans *Proc. Fourth Alvey Vision Conference*, p. 147–151.

Hartley, R. 1999, «Theory and Practice of Projective Rectification», *Intl. J. of Computer Vision*, vol. 35, p. 115–127.

Hartley, R. et A. Zisserman. 2003, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN 9780521540513. URL `http://books.google.fr/books?id=si3R3Pfa98QC`.

Herdt, A., H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur et M. Diehl. 2010, «Online Walking Motion Generation with Automatic Foot Step Placement», *Advanced Robotics*, vol. 24, nº 5-6, p. 719–737. URL `http://hal.inria.fr/inria-00391408/en`.

Hornung, A., K. Wurm et M. Bennewitz. 2010, «Humanoid Robot Localization in Complex Indoor Environments», dans *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, p. 1690–1695.

Jaillet, L., A. Yershova, S. La Valle et T. Simeon. 2005, «Adaptive tuning of the sampling domain for dynamic-domain rrts», dans *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, p. 2851–2856, doi :10.1109/IROS.2005.1545607.

Kaess, M., K. Ni et F. Dellaert. 2009, «Flow separation for fast and robust stereo odometry», dans *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Kobe, Japan.

Kajita, S., F. Kanehiro, K. Kaneko, K. Yokoi et H. Hirukawa. 2001, «The 3d linear inverted pendulum mode : a simple modeling for a biped walking pattern generation», dans *IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 239–246.

Kanehiro, F., W. Suleiman, F. Lamiraux, E. Yoshida et J.-P. Laumond. 2008, «Integrating dynamics into motion planning for humanoid robots», dans *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, p. 660 –667, doi :10.1109/IROS.2008.4650950.

Kaneko, K., F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi et T. Isozumi. 2004, «Humanoid robot hrp-2», dans *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 2, ISSN 1050-4729, p. 1083–1090 Vol.2, doi :10.1109/ROBOT.2004.1307969.

Kanoun, O. et J.-P. Laumond. 2010, «Optimizing the stepping of a humanoid robot for a sequence of tasks», dans *10th IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, p. 204–209, doi :10.1109/ICHR.2010.5686301.

Konolige, K. et M. Agrawal. 2008, «FrameSLAM : from bundle adjustment to real-time visual mapping», *IEEE Trans. Robotics*, vol. 24, n° 5, p. 1066–1077.

Kwak, N., O. Stasse, T. Foissotte et K. Yokoi. 2009, «3d grid and particle based slam for a humanoid robot», dans *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, IEEE, p. 62–67.

Laet, T. D., S. Bellens, R. S. andd Erwin Aertbeliën, H. Bruyninckx et J. D. Schuetter. 2012, «Geometric relation between rigid bodies : Semantics for standardization», *IEEE Robotics and Automation Magazine*.

Latombe, J.-C. 1991, *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA, USA, ISBN 079239206X.

LaValle, S. M. 2006, *Planning Algorithms*, Cambridge University Press, ISBN 0521862051. URL `http://www.worldcat.org/isbn/0521862051`.

Lefebvre, O., F. Lamiraux et D. Bonnafous. 2005, «Fast computation of robot-obstacle interactions in nonholonomic trajectory deformation», dans *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, p. 4612–4617.

Liskov, B. et J. M. Wing. 1994, «Family values : A behavioral notion of subtyping», cahier de recherche, ACM Transactions on Programming Languages and Systems.

de Luca, A., G. Oriolo et C. Samson. 1998, *Robot motion planning and control*, chap. Feedback Control of a Nonholonomic Car-like Robot, Jean-Paul Laumond éd., n° ISBN 3-540-76219-1 dans Lectures Notes in Control and Information Sciences 229, Springer.

Malgouyres, R. 2002, *Algorithmes pour la synthèse d'images et l'animation 3d*, Dunod.

Mansard, N., O. Stasse, P. Evrard et A. Kheddar. 2009, «A versatile generalized inverted kinematics implementation for collaborative working humanoid robots : The stack of tasks», dans *Proc. of the International Conference on Advanced Robotics (ICAR)*, p. 1–6.

Michel, P., J. Chestnutt, J. Kuffner et T. Kanade. 2005, «Vision-guided humanoid footstep planning for dynamic environments», dans *5th IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, p. 13–18, doi : 10.1109/ICHR.2005.1573538.

Miossec, S., K. Yokoi et A. Kheddar. 2006, «Development of a software for motion optimization of robots - application to the kick motion of the hrp-2 robot», dans *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on*, p. 299 –304, doi :10.1109/ROBIO.2006.340170.

Mombaur, K. 2009, «Using optimization to create self-stable human-like running», *Robotica*, vol. 27, n° 3, doi :10.1017/S0263574708004724, p. 321–330, ISSN 0263-5747. URL `http://dx.doi.org/10.1017/S0263574708004724`.

Mombaur, K., J.-P. Laumond et E. Yoshida. 2008, «An optimal control model unifying holonomic and nonholonomic walking», dans *8th IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, p. 646–653, doi : 10.1109/ICHR.2008.4756020.

Morisawa, M., K. Harada, S. Kajita, S. Nakaoka, K. Fujiwara, F. Kanehiro, K. Kaneko et H. Hirukawa. 2007, «Experimentation of humanoid walking allowing immediate modification of foot place based on analytical solution», dans *IEEE International Conference on Robotics and Automation (ICRA)*, ISSN 1050-4729, p. 3989 –3994, doi :10.1109/ROBOT.2007.364091.

Mouragnon, E., M. Lhuillier, M. Dhome, F. Dekeyser et P. Sayd. 2009, «Generic and Real-Time Structure from Motion using Local Bundle Adjustment», *Image and Vision Computing*, vol. 27, n° 8, p. 1178–1193.

Murray, R. M., Z. Li et S. S. Sastry. 1994, *A Mathematical Introduction to Robotic Manipulation*, CRC.

Muschevici, R., A. Potanin, E. Tempero et J. Noble. 2008, «Multiple dispatch in practice», dans *Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA '08, ACM, New York, NY, USA, ISBN 978-1-60558-215-3, p. 563–582, doi :10.1145/1449764.1449808. URL `http://doi.acm.org/10.1145/1449764.1449808`.

Nishiwaki, K. et S. Kagami. 2006, «High frequency walking pattern generation based on preview control of zmp», dans *IEEE International Conference on Robotics and Automation (ICRA)*, ISSN 1050-4729, p. 2667–2672, doi :10.1109/ROBOT.2006.1642104.

Nishiwaki, K., S. Kagami, J. Kuffner, K. Okada, Y. Kuniyoshi, M. Inaba et H. Inoue. 2002, «Online humanoid locomotion control by using 3d vision information», dans *Proc. of the IEEE/RSJ Int. Symposium on Experimental Robotics (ISER'02)*, vol. 99.

Nistér, D., O. Naroditsky et J. Bergen. 2004, «Visual Odometry», dans *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Washington, USA.

Oßwald, S., A. Hornung et M. Bennewitz. 2010, «Learning reliable and efficient navigation with a humanoid», dans *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, p. 2375–2380.

Ozawa, R., Y. Takaoka, Y. Kida, K. Nishiwaki, J. Chestnutt et J. Kuffner. 2005, «Using visual odometry to create 3D maps for online footstep planning», dans *IEEE Intl. Conference on Systems, Man and Cybernetics*, Hawaii, USA, p. 2643–2648.

Perrin, N., O. Stasse, F. Lamiraux et E. Yoshida. 2010, «Approximation of feasibility tests for reactive walk on hrp-2», dans *IEEE International Conference on Robotics and Automation (ICRA)*, ISSN 1050-4729, p. 4243–4248, doi :10.1109/ROBOT.2010.5509395.

Perrin N., Y. E., Lamiraux F. 2011, «A biped walking pattern generator based on "half-steps" for dimensionality reduction», dans *IEEE International Conference on Robotics and Automation (ICRA)*, p. 1270–1275.

Rives, P., F. Chaumette et B. Espiau. 1989, «Visual servoing based on a task function approach», dans *The First International Symposium on Experimental Robotics*, p. 412–428.

Samson, C. et K. Ait-Abderrahim. 1991, «Feedback control of a nonholonomic wheeled cart in cartesian space», dans *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, p. 1136–1141, doi :10.1109/ROBOT.1991.131748.

Sentis, L. et O. Khatib. 2005, «Control of free-floating humanoid robots through task prioritization», dans *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA*, p. 1718–1723.

Siciliano, B. et O. Khatib, éd.. 2008, *Springer Handbook of Robotics*, Springer, Berlin, Heidelberg, ISBN 978-3-540-23957-4. URL `http://dx.doi.org/10.1007/978-3-540-30301-5`.

Stasse, O., A. Davison, R. Sellaouti et K. Yokoi. 2006, «Real-time 3d slam for humanoid robot considering pattern generator information», dans *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Beijing, China, p. 348–355.

Stasse, O., T. Foissotte et A. Kheddar. 2008a, «Treasure hunting for humanoids robot», dans *IEEE RAS/RSJ International Conference on Humanoids Robots, Workshop on Cognitive Humanoid Vision (Daejeon, South Korea, 2008).*

Stasse, O., B. Verrelst, A. Davison, N. Mansard, F. Saidi, B. Vanderborght, C. Esteves et K. Yokoi. 2008b, «Integrating walking and vision to increase humanoid autonomy», *International Journal of Humanoid Robotics,special issue on Cognitive Humanoid Robots*, vol. 5, n° 2, p. accepted.

Stasse, O., B. Verrelst, P.-B. Wieber, B. V. P. Evrard, A. Kheddar et K. Yokoi. 2008c, «Modular architecture for humanoid walking pattern prototyping and experiments», *Advanced Robotics, Special Issue on Middleware for Robotics –Software and Hardware Module in Robotics System*, vol. 22, n° 6, p. 589–611.

Stoer, J. et R. Bulirsch. 2002, *Introduction to Numerical Analysis*, 3e éd., Springer, New York, ISBN 038795452X. URL `http://www.worldcat.org/isbn/038795452X`.

Strasdat, H., J. Montiel et A. Davison. 2010, «Real-time Monocular SLAM : Why Filter ?», dans *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, AK, USA, p. 2657–2664.

Suleiman, W., E. Yoshida, J.-P. Laumond et A. Monin. 2007, «On humanoid motion optimization», dans *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, p. 180 –187, doi :10.1109/ICHR.2007.4813866.

Tactical Technology Office. 2012, «Broad agency annoucement darpa robotics challenge», cahier de recherche, Defense Advanced Research Projects Agency (DARPA). DARPA-BAA-12-39.

Thompson, S., S. Kagami et K. Nishiwaki. 2006, «Localisation for autonomous humanoid navigation», dans *Proc. of the 6th IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, p. 13 –19, doi :10.1109/ICHR.2006.321357.

Thrun, S., W. Burgard et D. Fox. 2005, *Probabilistic Robotics*, Intelligent Robotics and Autonomous Agents, Mit Press, ISBN 9780262201629. URL `http://books.google.fr/books?id=k_yOQgAACAAJ`.

Vukobratovic, M. et B. Borovac. 2004, «Zero-moment point-thirty five years of its life», *International Journal of Humanoid Robotics*, vol. 1, n° 1, p. 157–173.

Whitehead, A. N. et B. Russel. 1910, *Principia Mathematica to *56*, Cambridge At The University Press.

Wikipedia. 2011, «Nao (robot) — wikipedia, the free encyclopedia», URL `http://en.wikipedia.org/wiki/Nao_(robot)`.

Yershova, A. et S. M. LaValle. 2007, «Improving motion-planning algorithms by efficient nearest-neighbor searching», *Robotics, IEEE Transactions on*, vol. 23, n° 1, doi :10.1109/TRO.2006.886840, p. 151–157, ISSN 1552-3098.

Yoshida, E., I. Belousov, C. Esteves et J.-P. Laumond. 2005, «Humanoid motion planning for dynamic tasks», dans *5th IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, p. 1–6, doi :10.1109/ICHR.2005.1573536.

Index